CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Game Design and Implementation of Physics-Based Interactions between Player and Environment Elements

Master's Thesis in Computer Science and Engineering

ADRIÁN NAVARRO PÉREZ

SAMUEL SOUTULLO SOBRAL

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Master's thesis 2020

Game Design and Implementation of Physics-Based Interactions between Player and Environment Elements

ADRIÁN NAVARRO PÉREZ
SAMUEL SOUTULLO SOBRAL

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

Game Design and Implementation of Physics-Based Interactions between Player and Environment Elements
ADRIÁN NAVARRO PÉREZ
SAMUEL SOUTULLO SOBRAL

Supervisor: Marco Fratarcangeli, Department of Computer Science and Engineering
Examiner: Staffan Björk, Department of Computer Science and Engineering

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2020

Game Design and Implementation of Physics-Based Interactions between Player and Environment Elements
ADRIÁN NAVARRO PÉREZ
SAMUEL SOUTULLO SOBRAL
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This master thesis presents the PEP framework, a formal method that guides the design and analysis of physics-based games, and *Under Surveillance*, a 2.5D physics-based puzzle adventure video game for PC.

The PEP framework provides a set of defined steps along with a common vocabulary all game designers can refer to when designing and analyzing physics-based games. The framework strives to ensure their consistency while simultaneously making more intuitive to the player the emerging behaviors consequence of these games' laws of physics.

In *Under Surveillance*, designed by applying the PEP framework and developed in Unity, players take control of a mysterious robot who can manipulate and make use of electricity and water to solve a set of puzzles as they progress further in a dystopian world.

Keywords: Game Design, Game Research, Game Development, Physics-Based Games, Philosophy, Psychology.

Acknowledgements

This master thesis was developed under the supervision and support of Marco Fratarcangeli. Adrián acknowledges the financial backing from *Fundación Margit y Folke Pehrzon*, *Svensk-Spanska Stiftelsen / The Swedish-Spanish Foundation* and *Adlerbertska Foreign Student Hospitality Foundation.* Samuel acknowledges the financial backing from *Fundación Barrié.*

Adrián Navarro Pérez and Samuel Soutullo Sobral, Gothenburg, June 2020

Contents

List of Figures

1 Introduction

1.1 Problem Description

Games are artifacts which support a voluntary interaction carried out, within a formal independent transmedial system, between one or more users and the system itself, performing a finite number of different types of actions without expecting a productive outcome [1–13].

Physics-based games are those in which purposeful gameplay is achieved by subjecting player's interactions or their consequences to a consistent set of laws of physics. All the occurrences, which rise as a consequence of gameplay, are ruled by these laws. Therefore, solid consistency in the emerging behaviors must be ensured.

Ensuring the consistency of these behaviors only adds more difficulties to the complex work of game design. Players need to comprehend how the laws of physics rule the environment from the game they are playing to make progress. Nevertheless, directly letting the player know about these laws and all their consequences is neither convenient nor practical.

1.2 Research Question

All the inconveniences presented in Sec. 1.1, naturally inherent to physics-based games, can be faced by making use of a common vocabulary and a formal method. By formalizing procedures to coordinate all working game designers during a project, it is ensured that a game is designed according to an established process in which a justifiable reason is found behind every single design decision. Thus, a more solid and consistent game design work is carried out, ultimately leading to the production of more successful products.

Hence, the research question this thesis pursues to answer is:

"What should be considered when designing physics-based interactions between player and environment elements to achieve more engaging experiences in physics-based games?"

1.3 Stakeholders

The needs of all the following involved stakeholders were taken into account.

1.3.1 Internal Stakeholders

The only internal stakeholders are the **master thesis authors** (Adrián Navarro Pérez and Samuel Soutullo Sobral). Both authors have a common interest in video game development. Thus, their personal and professional needs consisted of taking advantage of this project to improve their game development skills while breaking the bounds of their creativity. Their final goal was to create a new way to design more consistent and intuitive to the player physics-based games while developing a physics-based video game that matched their vision and expectations.

1.3.2 External Stakeholders

Since a stakeholder is any entity, including individuals and companies, that can affect or get affected by the project in any way, the following external stakeholders have been identified:

- **Game designers**: Game designers are in a permanent need of newer, easier and better ways to design games of every kind. Thus, one of the types of games they might design is physics-based games. Physics-based games do not rely on the same amount of resources and supportive tools to be designed like other types of games.
- **Players**: Players want a game that makes them feel in a very specific way. The game needed to be designed so it instantiated the proper aesthetics the master thesis authors wanted them to experience.
- **University of Gothenburg**: The university defined the formal requirements the thesis must fulfill. Moreover, it defines a set of deadlines and deliverables at the different stages of the master thesis development which had to be met by the master thesis authors so the thesis is academically successful.
- **Examiner (Staffan Björk)**: The role of the examiners is to ensure that the master thesis meets all the requirements previously set by the university to decide its final grade in the end.
- **Thesis supervisor (Marco Fratarcangeli)**: The role of the supervisor is to guide the students during their master thesis, to make sure that their work meets all the formal requirements given by the university and checked by the examiners. Therefore, he added limits to the project in certain situations, reducing its scope so the thesis could be academically successful.
- **Government and law**: The developed product had to be compliant with the pertinent legislation. This included, but it was not limited to copyright considerations, licensing and publishing.

1.4 Aim

The thesis aims to develop a formal process that guides the design and analysis of physics-based games, alleviating the challenges described in Sec. 1.1, faced by game designers when endeavoring for this task.

The work, far from setting a new standard in the game development industry, aspires to be seen as a new support tool that can be applied by game designers to facilitate their job in addition to already existing design methods [14, 15].

To verify the validity of the previous outcome, the formal process will be applied to design and analyze a physics-based game. The game will be fully developed to ensure the quality of the design.

2

Background

2.1 Game Design

Before developing any work related to the creation of the formal process to design and analyze physics-based games, research was conducted on other formal methods from the game design discipline to study the current state of the art.

2.1.1 MDA: A Formal Approach to Game Design and Game Research

The MDA framework defined in [15], which stands for Mechanics, Dynamics and Aesthetics, describes a formal approach to help designers, researchers and scholars to decompose, understand and create games.

As shown in Sec. 2.1, the MDA framework formalizes the consumption of games and its unpredictability by breaking them into the following three parts: Rules, Systems and "Fun".

Analogously, their three design counterparts are then established:

- **Mechanics** describes the particular components of the game, at the level of data representation and algorithms.
- **Dynamics** describes the run-time behavior of the mechanics acting on player inputs and each others' outputs over time.
- **Aesthetics** describes the desirable emotional responses evoked in the player, when she interacts with the game system, i.e., what makes a game "fun".

When developing games, both perspectives can be taken into consideration, both the player's and the designer's. This way of understanding games aims to help to comprehend how changes in one of the layers can affect the others.

However, the MDA framework does not take into account a deeper level of detail on the learning process of the player. Furthermore, this formal process does not ensure the consistency between the interactions that are held within the game environment and the events occurring during gameplay.

These two issues are important when designing physics-based games and proper relative gameplay experience. Thus, the need for a new formal process which would

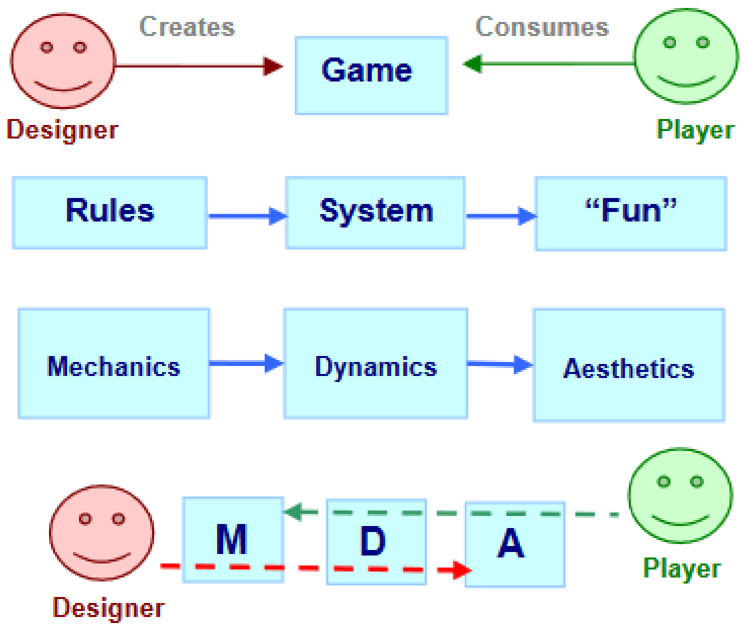


Figure 2.1: Formalization of the production and consumption of game artifacts. Source: [15].

take them into account and answer to the research question stated in Sec. 1.2 was emerging.

2.1.2 Other Methods

Other formal methods, such as [14], were studied but it was concluded that they were not sufficiently relevant to answer the research question. Therefore, the need for a new formal process that would take into account the consistency of physics-based games and how intuitive they should be for the player ultimately emerged.

2.2 Influential works

Video games are continuously influencing one another. They all take inspiration regarding what made a good game possible as they learn from what it did not quite work on them too.

Moreover, games are not the only source that they take inspiration from. Movies, books, music and even the mundane daily life from the world itself are set to study and observation. All this helps to better understand the surrounding world as well as the human behavior, helping game designers to better express and transmit their intentions towards each new game they design and develop.

To achieve a specific set of aesthetics to accomplish what the authors of the thesis envisioned, other video games, as well as a fair amount of books and movies, were taken into consideration. These are further explained below.

2.2.1 Lemmings

Lemmings [16] is a 2D puzzle-solving video game originally developed by DMA Design for the Amiga in 1991.

In each *Lemmings* level, the player must lead a group of lemmings from a starting point towards an exit gate while facing a sort of obstacles within a closed and manipulable environment. To avoid lemmings perishing from these obstacles or the environment itself, the player has access to a limited set of tools and actions which help lemmings to safely reach their goal. A couple of examples of these tools and actions are an umbrella which you can equip to lemmings, so they can safely land after a high fall, and making use of their own bare hands to excavate tunnels.

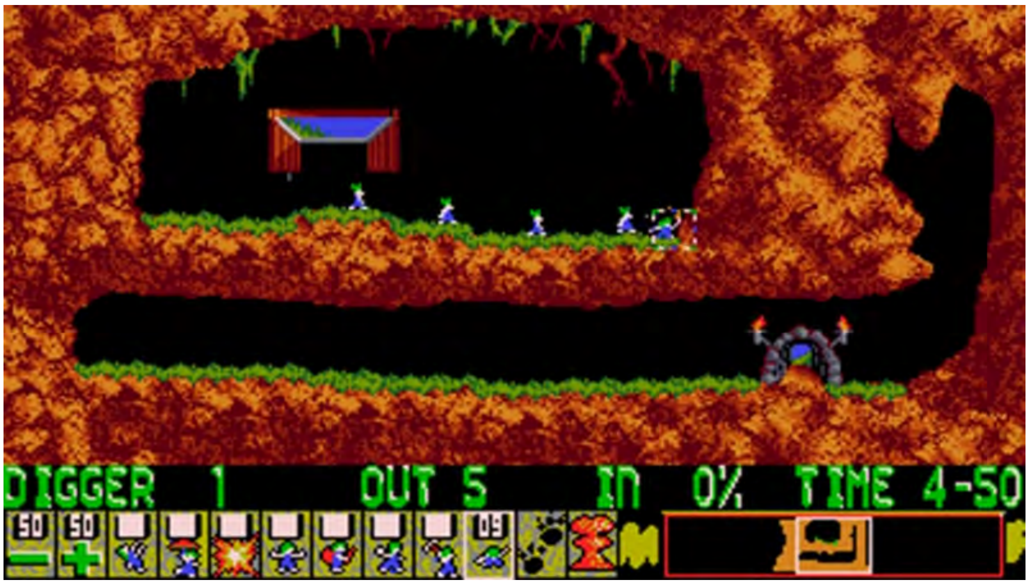


Figure 2.2: Lemmings first level in-game screenshot: A lemming is excavating a path towards the exit with its own bare hands after the player commanded it. Source: [17].

An interesting characteristic from *Lemmings* is that it belongs to the puzzle-solving genre while presenting 2D graphics. The authors of the thesis found this of interest, as according to their vision games that mix physics-based interactions with puzzle-solving mechanics have great potential to produce meaningful gameplay.

Furthermore, *Lemmings* effectively showcases all the information that must be available to the player at all times. This way, the challenge of solving each puzzle is the puzzle itself rather than other secondary factors such as relying on the player's memory. This was also in line with the vision of the authors.

Finally, in *Lemmings*, the animals always move automatically in two possible directions: left or right. Authors also found value in this feature, because it allows the player to focus on the puzzle-solving mechanics.

2.2.2 Limbo

Limbo [18] is a 2D puzzle-solving platformer video game developed by Playdead and first released for Xbox Live Arcade in July 2010.

In *Limbo*, the player takes control of a mysterious boy who progresses through an unsafe, dark and horrific environment full of dangerous and hazardous elements to search for his sister. The boy faces an unnumbered amount of deathly situations often given when solving incorrectly one of the game's puzzles.

One of the best points of *Limbo* is its unique visual style and its eerie atmosphere complemented by a sound design commonly found only in horror and thriller movies. This was precisely the set of strong points that inspired the master thesis authors to design the aesthetics of their game.

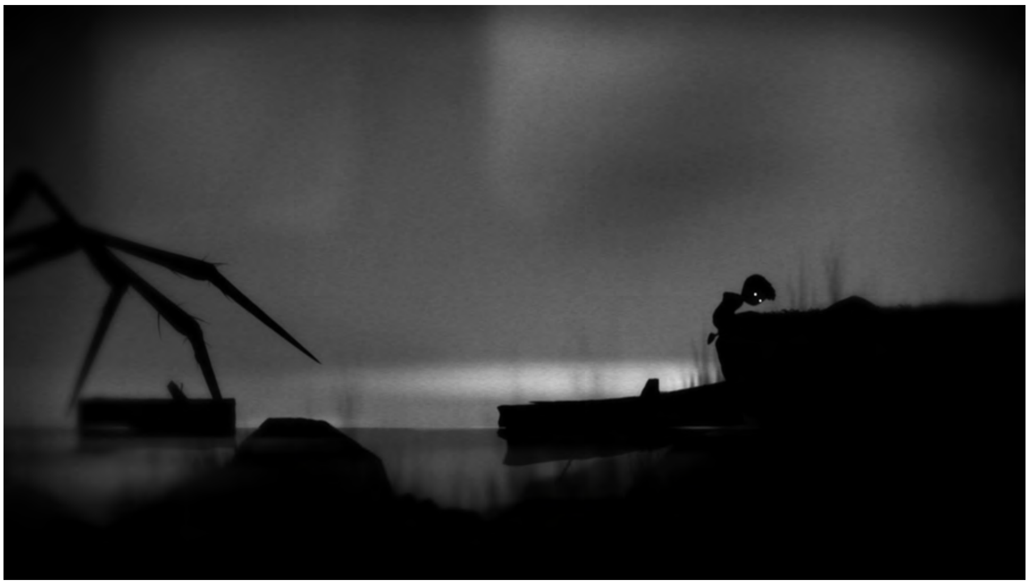


Figure 2.3: Limbo in-game screenshot: The boy is fleeing away from a giant spider. Source: [19].

2.2.3 Inside

Inside [20] is a 2.5D puzzle-solving platformer adventure video game brought first to Xbox One in June 2016 by the same developers of *Limbo*, Playdead.

In *Inside*, the player takes control of a boy who lives in a dystopian world. Along the way, the boy solves puzzles while avoiding death.

Following the steps of its predecessor, the art direction of *Inside* is magnificent to recreate the dystopian world where the game takes place. Unlike Limbo, *Inside* makes use of 3D models but restricting the movement of the character to only one plane, which leads to the development of a 2.5D side-scrolling game.

The master thesis authors found particularly interesting *Inside*'s approach of presenting 2.5D graphics. It was concluded that this style provides a good balance between development effort and design possibilities.

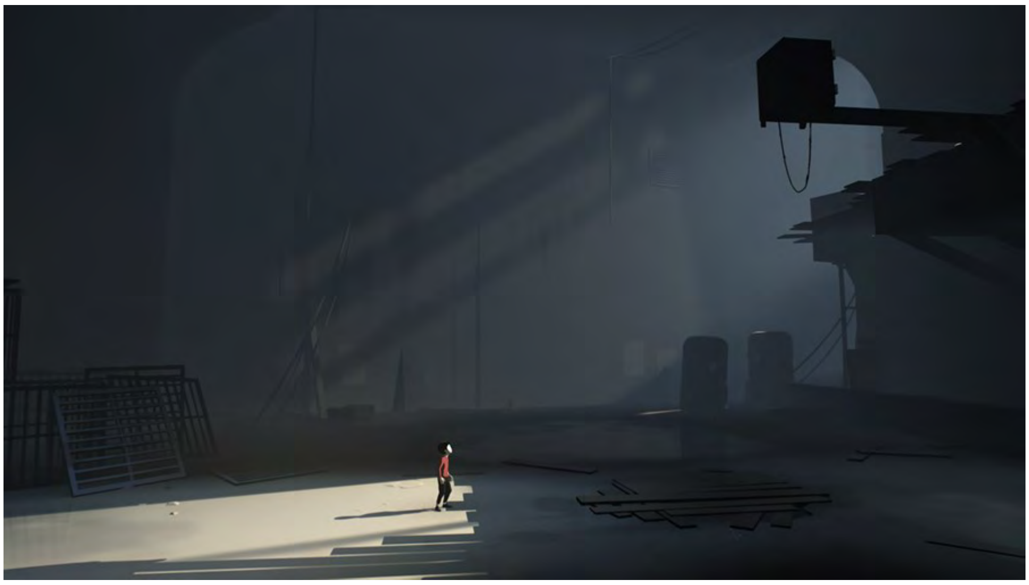


Figure 2.4: Inside in-game screenshot: The boy is looking at a box located close to the ceiling. Source: [19].

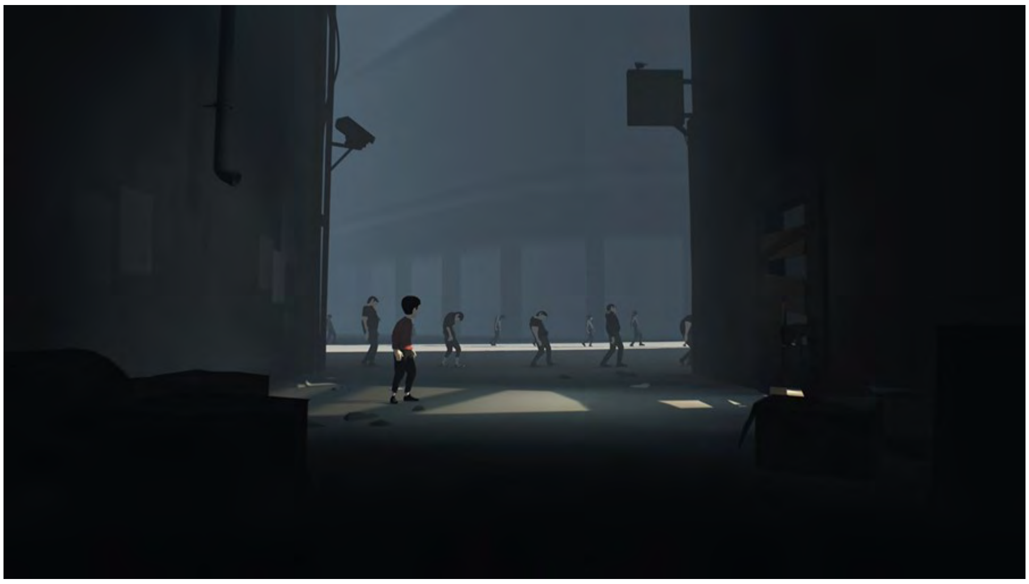


Figure 2.5: Inside in-game screenshot: The boy is looking at a group of people walking over the street. Source: [19].

2.2.4 Nineteen Eighty-Four (1984)

Nineteen Eighty-Four: A Novel, generally published as *1984* is an English novel written by Eric Arthur Blair, better known by his pen name, George Orwell, published in 1949.

Nineteen Eighty-Four: A Novel follows the story of Winston Smith, a skeptic middle-age man living under a totalitarian government based on the mass surveillance of the entire population. The figurehead and leader of the party who rules the government is represented as a person with a mustache known as Big Brother.

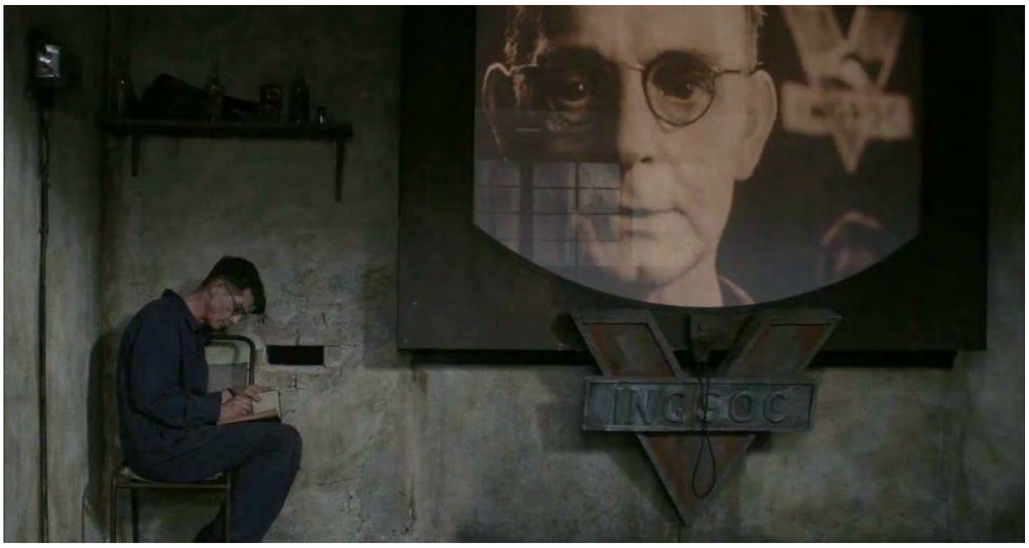


Figure 2.6: Shot from 1984 (Nineteen Eighty-Four), the movie adaption of *Nineteen Eighty-Four: A Novel*. Source: [21].

The narrative George Orwell builds around a mass surveillance state in a dystopian world was found of great interest.

2.2.5 Blade Runner

Blade Runner is the science fiction film released in 1982 based on the book *Do Androids Dream of Electric Sheep?* written by Philip Kindred Dick.

Blade Runner narrates the story of a retired cop from Los Angeles called Rick Deckard. In 2019, Los Angeles hosts a cyberpunk and dystopian society where replicants, human-like androids produced by the Tyrell Corporation to colonize dangerous off-world locations, coexist with the humankind. Rick Deckard was a *blade runner* entrusted to track down and eliminate rogue replicants. Suddenly, he needs to get back to his job to stop a group of replicants who have escaped and headed to Earth, assassinating many humans on their way.

The futuristic cyberpunk dystopian society is perfectly showcased by the magnificent look of the film. High contrast between wealth and poverty derived from this kind of world is found. This contrast is supported by the numerous skyscrapers full of dotted lighting and the flying vehicles, *spinners*, navigating between them. On the other hand, the streets located at the bottom of the same skyscrapers are full of humming and rumbling sounds and gloomy lighting as a result of all the neon lights placed.

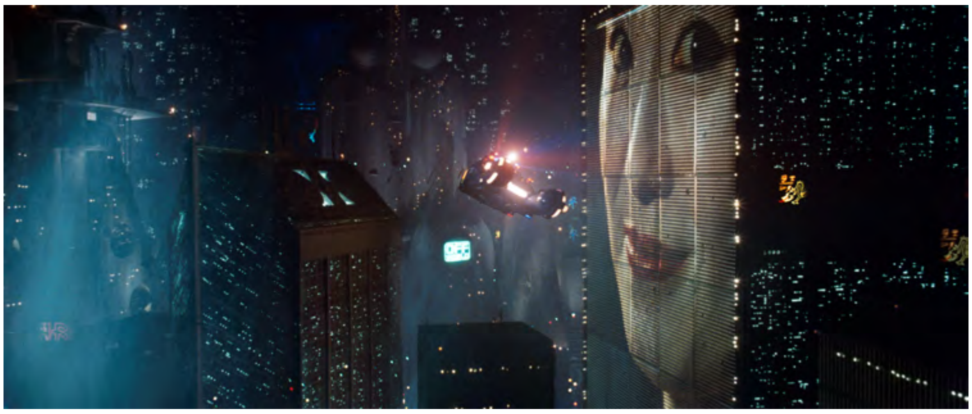


Figure 2.7: Skyscrapers from Los Angeles during the night. Source: [22].

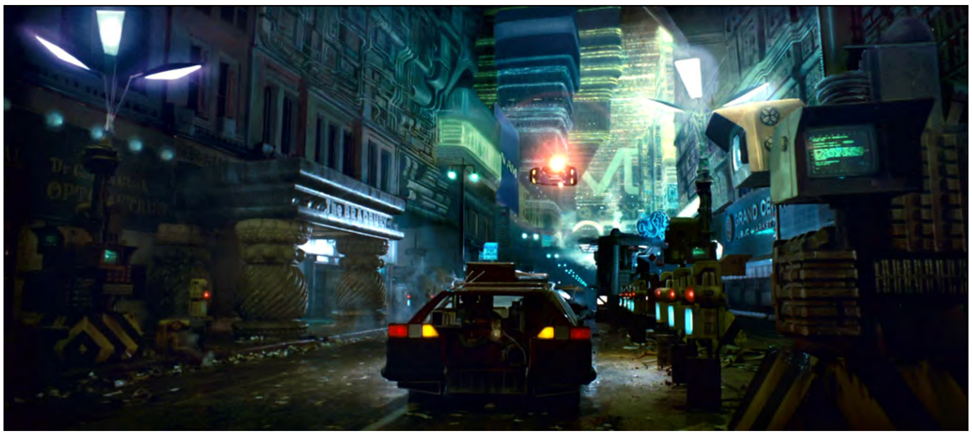


Figure 2.8: Streets from Los Angeles during the night. Source: [22].

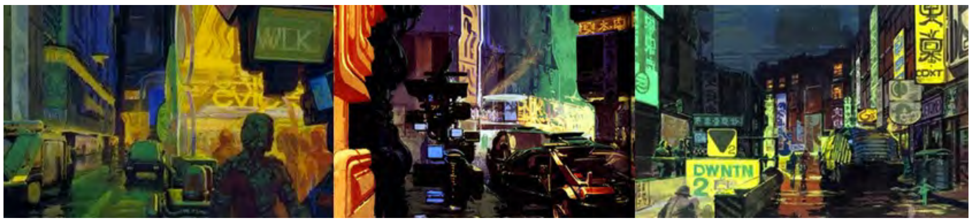


Figure 2.9: Syd Mead's *Blade Runner* concept art. Source: [22].

The master thesis authors concluded that a combination of the aforementioned type of atmospheres found in *Limbo* and *Inside* in combination with *Blade Runner*'s thematic design, could provide the resulting game with a unique visual style.

3
Theory

3.1 Philosophy

As the starting point to develop the envisioned formal process to design and analyze physics-based games, research was conducted in the field of philosophy, in particular on the most relevant epistemological currents. From an epistemological perspective, it was found a correlation between how the real world and physics-based games work. They both host emerging events according to the laws of physics. Additionally, the humankind learns through the observation of such events in both scenarios indistinctly. For example, humans learned about gravity because they were able to perceive its effects and impact on the surroundings.

The *Allegory of the Cave* by Plato, part of the Book VII from his Socratic dialogue, The Republic [23], and the *Critique of the Pure Reason* by Kant [24] influenced the thesis work.

3.1.1 The Allegory of the Cave

The *Allegory of the Cave* is one widely-known allegory which explains the pillars of Platonism in a simple but profound way. It explains human perception concerning true knowledge and how to gain it over philosophical reasoning.

In the *Allegory of the Cave*, Plato describes a cave inhabited by a group of people who are prisoners since they were born and can only see the walls of the cave. Behind them, another wall and a corridor they cannot see are located. The corridor is populated by another group of people who are walking and carrying around a set of objects. Behind, and hidden to the prisoners, there is also a bonfire enlightening the corridor and illuminating the wall of the cave the prisoners see. Thus, the prisoners observe the shadows projected on the wall by the objects the other group of people carries around.

The projected shadows become the only possible truth for the prisoners. They are not aware of what is happening out of their understanding. For them, the real world is limited to those shadows, which in reality are just a small consequence of a greater whole: a world where light interacts with opaque objects generating shadows.

Fig. 3.1 illustrates how the most important elements from the *Allegory of the Cave* relate to one another. This diagram synthesizes the parable to what is relevant for the thesis and its development. In Sec. 6.2.1 how the concepts are related to the framework is explained.

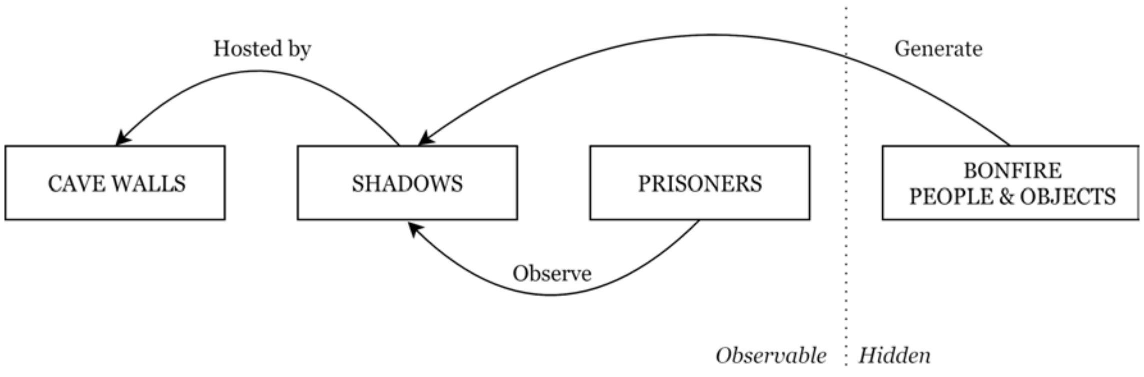


Figure 3.1: Conceptual representation of the *Allegory of the Cave.*

3.1.2 Critique of Pure Reason

Critique of Pure Reason, first published in 1781, is the work in which Immanuel Kant explores the boundaries of metaphysics. In this work, some of the most significant developed concepts become of great influence in the thesis work. These concepts and the relations between them are represented in Fig. 3.2.

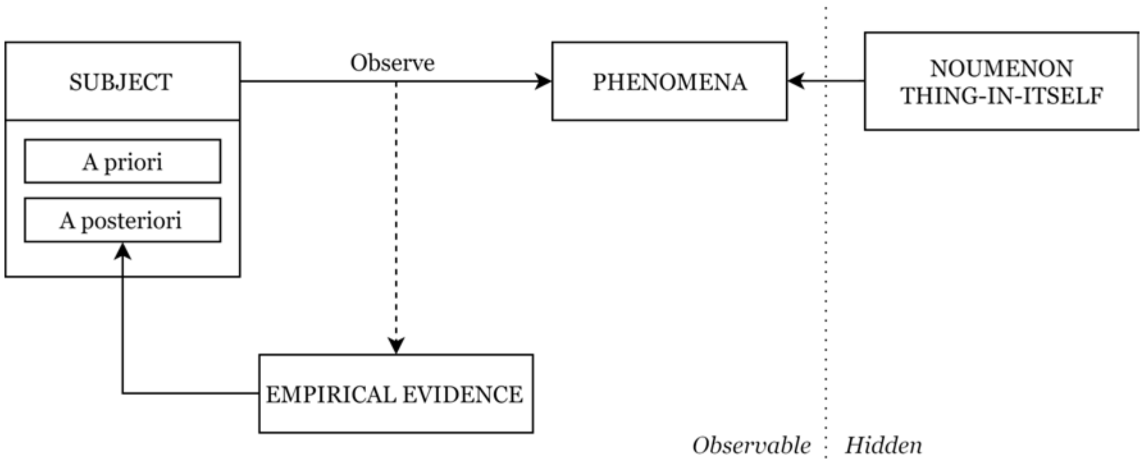


Figure 3.2: Conceptual representation of the relevant concepts from the *Critique of Pure Reason.*

Kant refers to *a priori* as everything transcendental, i.e., everything related to knowledge previous to the experience, such as time, space and fundamentals of logic. This kind of knowledge is inherent to the subject and it cannot be used to acquire new knowledge just by itself.

On the other hand, the knowledge derived from the experience, often providing novel knowledge to the subject, is known as *a posteriori.* This knowledge is acquired and verified through *empirical evidence* via observations. Nevertheless, *a posteriori* knowledge is not universal. It may be false in any case other than the observed.

Kant also argues that reality in itself is unknown to subjects. The *thing-in-itself*, or *noumenon*, is inaccessible to them, hence it cannot be perceived by the subjects. Consequently, the reality can only be perceived through *phenomena*, which is how it subjectively appears to them. According to Kant, a better conception of reality would not be how it is, but how it is perceived by a specific subject. Understanding reality implies shaping it with the subject's *a priori* knowledge.

3.2 Psychology

The research was also conducted in the field of psychology with a strong focus placed on the foundations of cognition, the reason being that the thesis work intends to take into consideration the player's learning process during gameplay. Two close-related psychological concepts, namely, *mental model* and *double-loop learning*, became indispensable when developing the formal process to design and analyze physics-based games.

3.2.1 Mental Model and Double-Loop Learning

A *mental model* is a user's internal and structured representation of a system. It is originated or modified from the interaction between the user and external events which help to guide the user's actions and to interpret the system's behavior. [25–28].

When users address a specific goal, the *mental model* might need to be modified to gain a reactive and deep understanding of their surroundings to accomplish such goal. This modification relies upon the user's individual experience (reflexive thinking). This cognitive process is known as *double-loop learning* [29].

In Fig. 3.3, *double-loop learning* is illustrated by representing a user observing its surroundings while gathering information. This information is then used to update their *mental model*, i.e., their interpretation of the real world. All the other elements show all the information being processed by the user, ultimately used to make decisions and accomplish goals that may have an impact on the world.

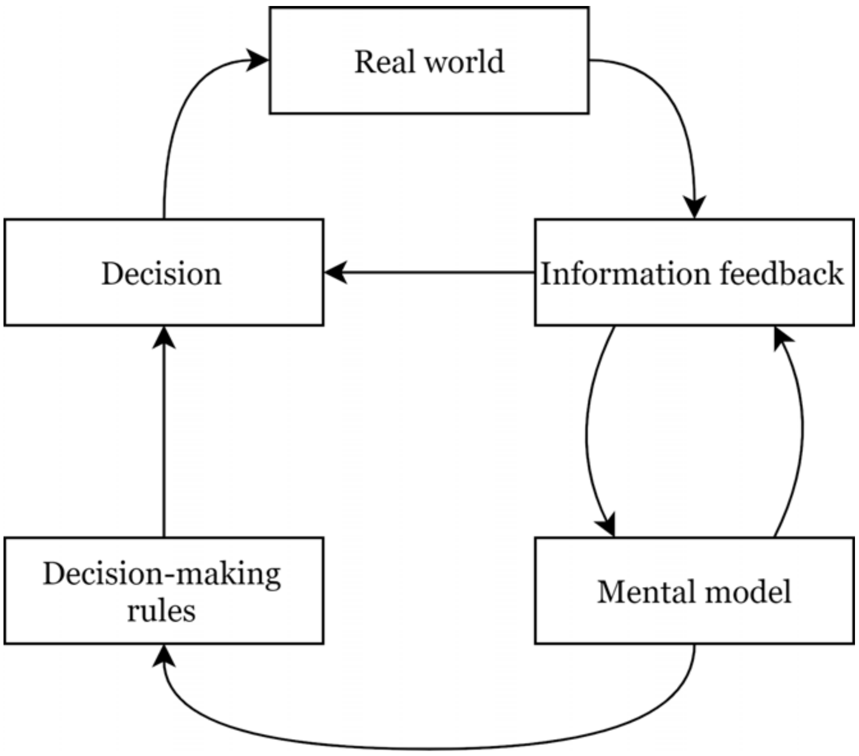


Figure 3.3: Conceptual representation of double-loop learning. Adapted from [30].

4

Methodology

4.1 Agile Development Methodology

The development methodology employed to develop the work of the thesis was highly based on *Scrum*, an agile software development framework.

Agile software development comprises a set of principles, practices and methodologies that guide software development processes. These are based on having an iterative development process, in which there are no big upfront requirements nor software architecture. Instead, a continuous evolution process is held. This kind of development is performed in dedicated, small, cross-functional teams selecting work, from one prioritized backlog, which gets tested regularly [31].

The compendium of ideas that *Agile* manifests is applied by several other methodologies, such as *Scrum*. In *Scrum*, every task is organized in a prioritized backlog and completed during the *Sprints*. *Sprints* are time-boxed iterations in which a set of tasks directly selected from the backlog are meant to be completed by the end of each iteration. Fig. 4.1 illustrates the workflow on *Scrum* projects, such as the one carried out in this thesis [31].

Before the beginning of each *Sprint*, estimation on the temporal cost of each task must be performed. These estimations aid the planning of which tasks should be included in each *Sprint*. Nonetheless, *Sprints* must be relatively short, usually no more than a month, so the execution process does not deviate from the plan over long periods.

Another key element from *Scrum* are the meetings. In particular, there are three different types of meetings: daily *Scrum* meetings, *Sprint* reviews and *Sprint* retrospectives. The purpose of daily meetings is to coordinate the team for the next 24 hours and detect any potential problems that might compromise the *Sprint*. *Sprint* reviews consist on discussing the results of each *Sprint* at the end of them. *Sprint* retrospectives take place after the *Sprint* reviews and their objective is to analyze what was correctly and incorrectly done during the duration of the *Sprint* and how to improve the development process based on these observations.

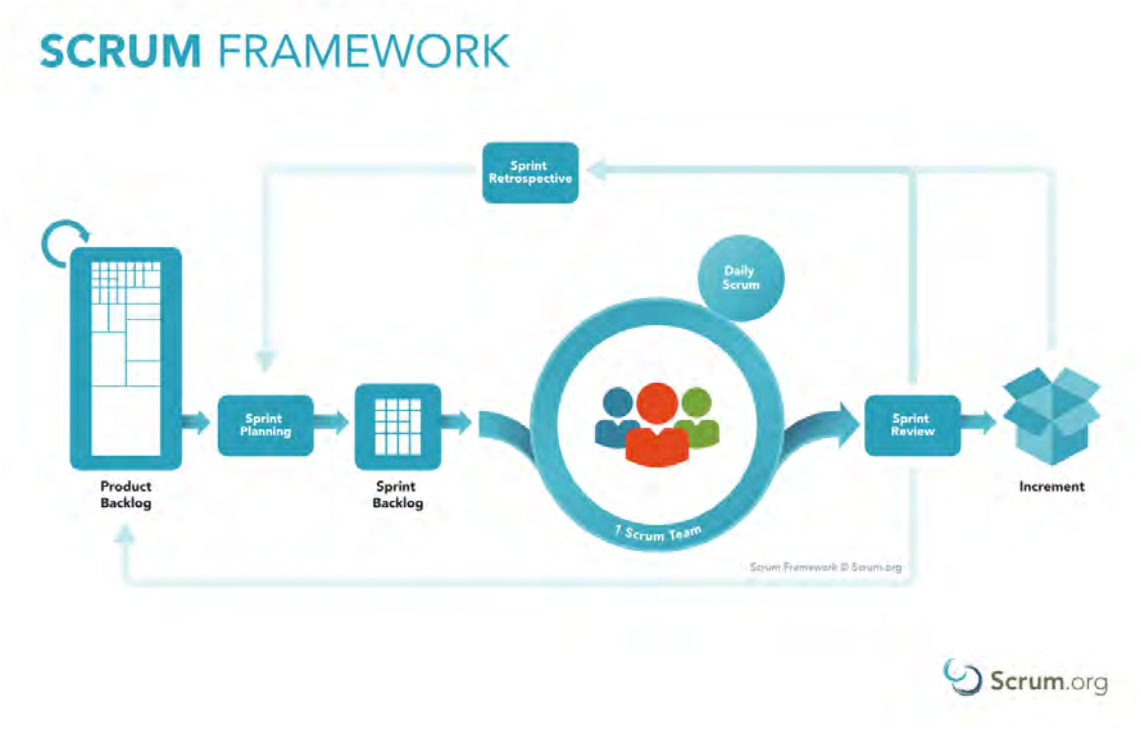


Figure 4.1: Scrum workflow. Source: [32].

4.1.1 Applied Principles, Practices and Methods

No Big Upfront Decisions

Having an iterative process with no big upfront requirements was the core of the whole process. The initial requirements consisted solely on creating a physics-based game in which a character could use electricity, water and fire. These requirements would then be further refined in an iterative process in which intermediate prototypes of the game were created, modified and polished until a satisfactory result was achieved.

To compensate for the lack of an initially defined software architecture of the video game, refactoring was essential in its development.

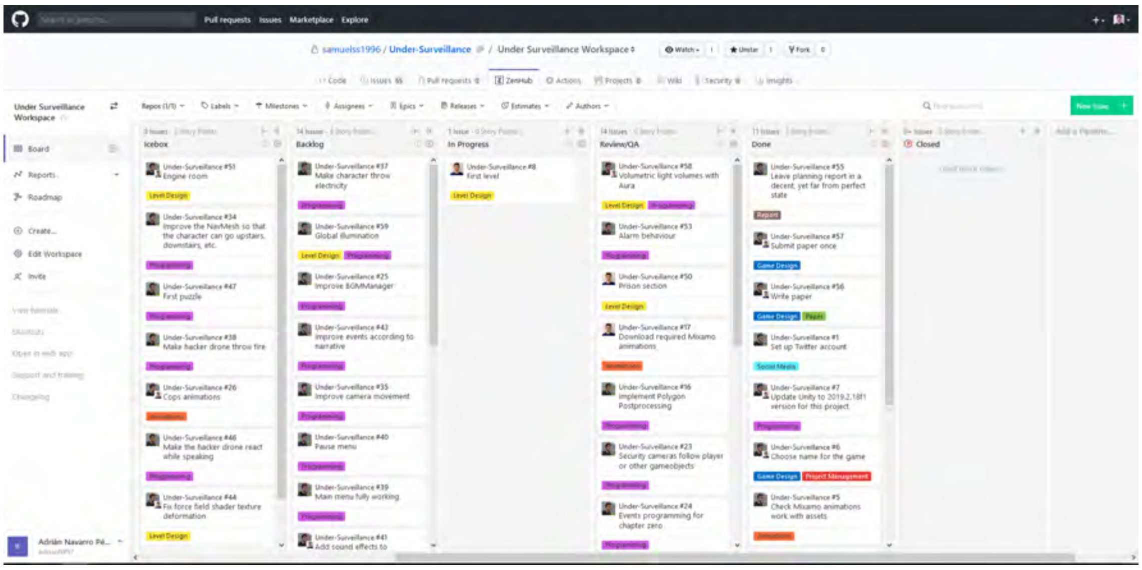
Meetings

Concerning the meetings carried out by the master thesis authors, these did not take place according to a fixed regular schedule. The small team size was one of the reasons behind this decision. This condition made possible the use of more informal channels to continuously exchange information on the current status of the project that ultimately made meetings less necessary. Nonetheless, when it was required to have complex discussions or take intricate decisions, meetings took place.

In addition to the meetings between the master thesis authors, periodic meetings were held between the two and the master thesis supervisor to discuss even more complex decisions and set the scope of the master thesis.

Prioritized Backlog

About team and tasks organization, a prioritized backlog was used. In particular, *ZenHub* [33] was the tool used for this purpose. *ZenHub* is an agile project management *GitHub* [34] plugin that provides a way to organize issues into tasks, sorted in different categories. Fig. 4.2 shows one screenshot from the *ZenHub* backlog of the project.



Icebox
- Under-Surveillance #51 Engine room — Level Design
- Under-Surveillance #34 Improve the NavMesh so that the character can go upstairs, downstairs, etc. — Programming
- Under-Surveillance #47 First puzzle — Programming
- Under-Surveillance #38 Make hacker drone throw fire — Programming
- Under-Surveillance #26 Cops animations — Animations
- Under-Surveillance #46 Make the hacker drone react while speaking — Programming
- Under-Surveillance #44 Fix force field shader texture deformation — Level Design

Backlog
- Under-Surveillance #37 Make character throw electricity — Programming
- Under-Surveillance #59 Global illumination — Level Design, Programming
- Under-Surveillance #25 Improve BGMManager — Programming
- Under-Surveillance #43 Improve events according to narrative — Programming
- Under-Surveillance #35 Improve camera movement — Programming
- Under-Surveillance #40 Pause menu — Programming
- Under-Surveillance #39 Main menu fully working — Programming
- Under-Surveillance #41 Add sound effects to

In Progress
- Under-Surveillance #8 First level — Level Design

Review/QA
- Under-Surveillance #58 Volumetric light volumes with Aura — Level Design, Programming
- Under-Surveillance #53 Alarm behaviour — Programming
- Under-Surveillance #50 Prison section — Level Design
- Under-Surveillance #17 Download required Mixamo animations — Animations
- Under-Surveillance #16 Implement Polygon Postprocessing — Programming
- Under-Surveillance #23 Security cameras follow player or other gameobjects — Programming
- Under-Surveillance #24 Events programming for chapter zero — Programming

Done
- Under-Surveillance #55 Leave planning report in a decent, yet far from perfect state — Report
- Under-Surveillance #57 Submit paper once — Game Design
- Under-Surveillance #56 Write paper — Game Design, Paper
- Under-Surveillance #1 Set up Twitter account — Social Media
- Under-Surveillance #7 Update Unity to 2019.2.18f1 version for this project — Programming
- Under-Surveillance #6 Choose name for the game — Game Design, Project Management
- Under-Surveillance #5 Check Mixamo animations work with assets — Animations

Closed

Figure 4.2: *Project*'s backlog.

As can be seen, there is a formal structure that helps to organize and keep track of all the status of the tasks. In particular, all the tasks are organized in specific categories and are defined according to a given structure.

The categories are:

- **Icebox**: This category contains tasks that are currently not part of the scope of the project but might be in the future. If at any point given the team considers that a task placed in this category is worth implementing, it is then moved to the *backlog* category.
- **Backlog**: This category contains all the tasks that are currently part of the scope of the project, meaning that it is planned that they are implemented. These tasks are sorted by priority, i.e., the task in the top must be implemented first, then one below it, and so forth. When a task from this category starts to be developed by the team, it is moved to the *in progress* category.
- **In Progress**: This category contains all the tasks that the team has started to develop but are not finished yet. When a task is finished, it is moved to the *Review/QA* category.
- **Review/QA**: This category contains all the tasks that are finalized, but are still pending from verification by both team members as well as deeper testing. When both team members agree that the task has been accomplished as

expected both in terms of functionality and quality, it is then moved to the *done* category.

- **Done**: This category contains all the tasks that have been completely developed, including verification and testing.

The fields that form each task are:

- **Name**: A brief name that helps to quickly identify the task.
- **Description**: One or more paragraphs that provide detailed information on what the task consists. This field may contain a list of subtasks that must be completed in order to carry out the task in question.
- **Assignees**: A list of the members of the project that are responsible for developing the task. To decide to whom each task is assigned, both members of the project discuss and reach an agreement based on various factors, such as their current workload, their level of specialization for the task, etc.
- **Labels**: A list of one or more specific tags that help to identify the nature of the task. The tags are: *animations*, *game design*, *level design*, *narrative design*, *paper*, *programming*, *project management*, *report*, *social media* and *sound design*.

As an example, Fig. 4.3 shows one of the many tasks present in the project's *ZenHub* with all the previously explained fields filled.

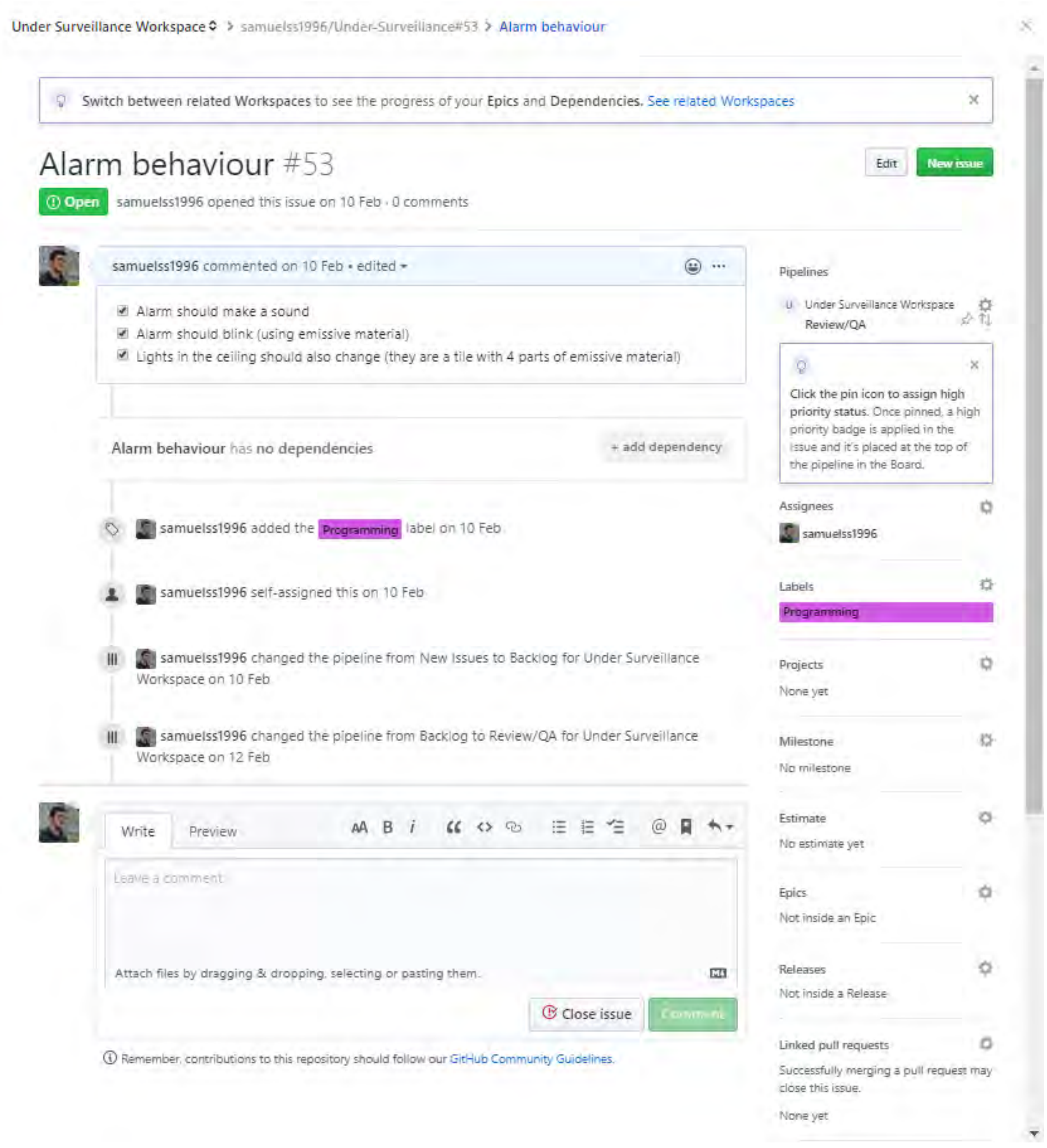
Project Diary

As it is described in [35], it is very beneficial to write a brief diary or journal of every project. The intention behind this document is not to write something worth reading for, but rather to note every significant piece of information, event and all the problems and solutions that are coming along the life cycle of the project. As the diary keeps being filled with project specifications, meetings' minutes and other types of relevant information, this document becomes a valuable item both for the present and the future of the project as it progresses.

Keeping diaries from previous projects can also help on how to approach upcoming ones. Storing information about past mistakes and how the writer of the diary tried to solve them, regardless of their success, is a very relevant lesson to take into account when heading towards the development of new projects.

Nowadays, project diaries can be stored online so multiple project managers can have immediate access to them to write new chunks of information or make modifications on the go. This greatly improves the usage speed of this methodology overall.

In the case of this master thesis, a *Google Docs* document was created and shared with all the master thesis authors and the supervisor in *Google Drive* at the beginning of the project. Therefore, the diary was available to everyone at all times during the whole life cycle of the master thesis.


Under Surveillance Workspace > samuelss1996/Under-Surveillance#53 > Alarm behaviour

Switch between related Workspaces to see the progress of your Epics and Dependencies. See related Workspaces

Alarm behaviour #53

Open samuelss1996 opened this issue on 10 Feb · 0 comments

Edit New issue

samuelss1996 commented on 10 Feb • edited

- [x] Alarm should make a sound
- [x] Alarm should blink (using emissive material)
- [x] Lights in the ceiling should also change (they are a tile with 4 parts of emissive material)

Alarm behaviour has no dependencies + add dependency

samuelss1996 added the Programming label on 10 Feb

samuelss1996 self-assigned this on 10 Feb

samuelss1996 changed the pipeline from New Issues to Backlog for Under Surveillance Workspace on 10 Feb

samuelss1996 changed the pipeline from Backlog to Review/QA for Under Surveillance Workspace on 12 Feb

Write Preview

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Close issue Comment

Remember, contributions to this repository should follow our GitHub Community Guidelines.

Pipelines

Under Surveillance Workspace
Review/QA

Click the pin icon to assign high priority status. Once pinned, a high priority badge is applied in the issue and it's placed at the top of the pipeline in the Board.

Assignees
samuelss1996

Labels
Programming

Projects
None yet

Milestone
No milestone

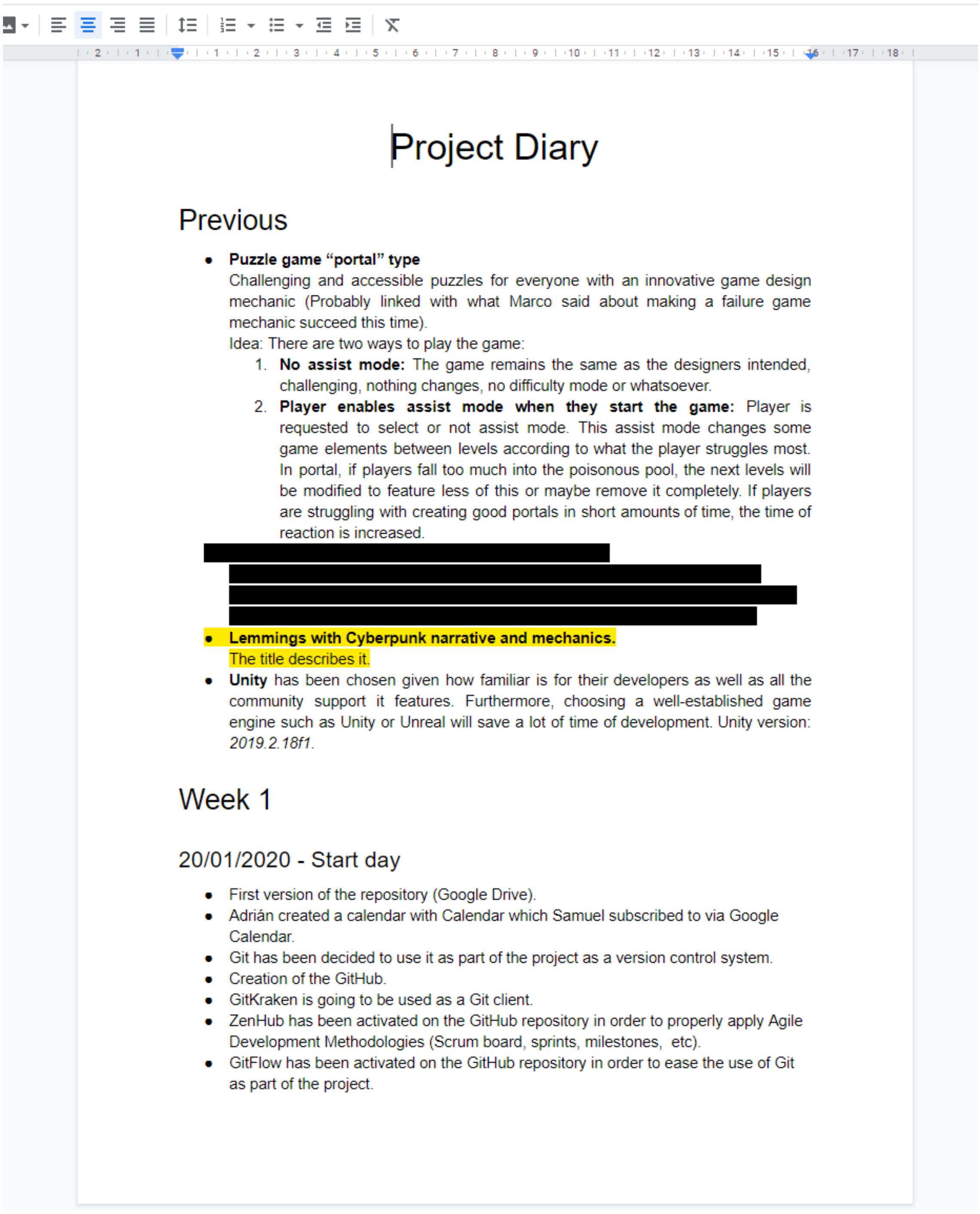
Estimate
No estimate yet

Epics
Not inside an Epic

Releases
Not inside a Release

Linked pull requests
Successfully merging a pull request may close this issue.
None yet

Figure 4.3: One of the tasks present in the project's *ZenHub*.



Project Diary

Previous

- **Puzzle game "portal" type**
  Challenging and accessible puzzles for everyone with an innovative game design mechanic (Probably linked with what Marco said about making a failure game mechanic succeed this time).
  Idea: There are two ways to play the game:
  1. **No assist mode:** The game remains the same as the designers intended, challenging, nothing changes, no difficulty mode or whatsoever.
  2. **Player enables assist mode when they start the game:** Player is requested to select or not assist mode. This assist mode changes some game elements between levels according to what the player struggles most. In portal, if players fall too much into the poisonous pool, the next levels will be modified to feature less of this or maybe remove it completely. If players are struggling with creating good portals in short amounts of time, the time of reaction is increased.

[redacted]

- **Lemmings with Cyberpunk narrative and mechanics.**
  The title describes it.
- **Unity** has been chosen given how familiar is for their developers as well as all the community support it features. Furthermore, choosing a well-established game engine such as Unity or Unreal will save a lot of time of development. Unity version: *2019.2.18f1*.

Week 1

20/01/2020 - Start day

- First version of the repository (Google Drive).
- Adrián created a calendar with Calendar which Samuel subscribed to via Google Calendar.
- Git has been decided to use it as part of the project as a version control system.
- Creation of the GitHub.
- GitKraken is going to be used as a Git client.
- ZenHub has been activated on the GitHub repository in order to properly apply Agile Development Methodologies (Scrum board, sprints, milestones, etc).
- GitFlow has been activated on the GitHub repository in order to ease the use of Git as part of the project.

Figure 4.4: First page out of thirty-eight of the master thesis' project diary.

Embrace the Change

In the second principle[1] in the Agile Manifesto [37], embracing all change is promoted.

Embracing the change is indispensable within every project regardless of its nature. Many developers might get personally attached to their work during the project. An example of this applied to game development is found in level design. Level designers often spend a lot of time designing new levels and new ways of playing the game. Therefore, when a level needs to be discarded for any reason being, if the level designer is not able to accept the change and embrace it, it can lead to a situation where they start defending that the level is necessary when it might be not. This kind of troubles often means a loss of resources either in terms of money or time.

Continuous Refactoring

It is understood by refactoring [31] as the technical process which examines the current design or implementation of units of the software to improve it without altering its output.

The objective behind this practice is to improve the consistency of the units while maintaining a clean design and adjusting the code to any applicable coding standards, further explained in Sec. 4.1.1.

In the game developed throughout the thesis, a lot of code was continuously added to the game which was naturally and progressively growing messier. If the technical process of refactoring was not periodically applied to such code, eventually it would have meant a big loss of time given the situation.

Pair Programming

Pair programming [31] is a technical practice in which code is systematically developed by two programmers, often closely involved in the work, sharing and using only one workstation. The first one of them takes control of the keyboard and mouse while thinking aloud every single decision taken. On the other hand, the second person stays next to the first one commenting, criticizing and making suggestions on the work the first one is carrying out.

As this practice is a peer process, the co-workers should reverse roles regularly.

The goal behind this practice is to force the person using the workstation to explain their thinking aloud to the second one, often both realizing right away about the mistakes they might be making. This way, mistakes usually hindered by overlooking, are easier to detect.

In the master thesis, the amount of time was very limited. Therefore pair programming was a crucial practice to minimize the loss of time searching for small mistakes

[1]Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage [36].

and solving them.

Coding Standards

It is understood by coding standards [31] as the defined style rules that agile teams should apply to all the code they produce to keep a general consistency over all the units of the software.

Coding standards are defined and shared with all the development teams early in the development of the project so the aforementioned consistency is kept from the very early stages of the project's development.

Although this master thesis was not a very large project in terms of software, defining early coding standards helped to quickly understand the different units of the software regardless of who had developed them.

Shared Code Ownership

In many development groups, each of the units of the software usually belongs to an individual, often their creator, meaning that they are the people in charge of the units. In other words, they need to personally take part in every single decision regarding such units, leading to slower decision making.

Agile methods are instead supporting shared code ownership [31]. In development groups in which shared code ownership is supported, the whole development team is responsible for all of the code. This way, the dependence on individuals is removed, and team members grow a more personal approach towards the code. Moreover, this all translates to the possibility of making new changes into the units of the software, even if they will affect several parts of the system, without consulting every single decision beforehand.

In this master thesis, the usage of *GitFlow* as described in Sec. 4.1.2 was of great aid towards the implementation of shared code ownership within the project.

4.1.2 Version Control Methodology: *GitFlow*

Version control is a crucial aspect of software development. Version control tools, such as *Git* [38], allow keeping an organized database of all the early, intermediate and final versions of any software.

GitFlow [39] is a branching model that aims to organize software development and coordinate teamwork.

In *GitFlow*, there are two main branches:

- `master`: The main branch into which any version committed must be ready for production.
- `develop`: All the intermediate versions created between each production versions are committed to this branch.

The workflow consists on working on the `develop` branch, committing intermediate versions until a production version is achieved. Then, `develop` is merged into `master`, thus finalizing the process required to create a new production version.

Furthermore, there are also three different types of auxiliary branches:

- `feature`: This type of branches always originates from and is merged into `develop`. They are used to develop new features of the software, that, when finalized, are incorporated into `develop`.
- `release`: This type of branches originate from `develop` and are merged both into `develop` and `master`. The purpose of these branches is to perform small changes and adjustments required in a production version before releasing it. It is after merging a branch of this type into `master` that the production version is ready to be distributed.
- `hotfix`: This type of branches originate from `master` and are merged both into `master` and `develop`. The purpose of these branches is to fix bugs in production versions. They are created from a version present in `master`. Then, the required fixes are committed to the `hotfix` branch. Finally, the changes are incorporated both to `master` and `develop`. Thus, a new production version with the problem fixed is immediately created, while also the fix is kept for future versions.

In order to keep the repository organized and being able to quickly identify to which category a particular branch belongs, it is important to follow a specific naming convention that solves this issue. A commonly used approach is naming `feature` branches starting with *feature/*, `release` branches starting with *release/* and `hotfix` branches starting with *hotfix/*.

In order to ensure that the *GitFlow* branching model was followed, the management of the branches was not performed manually, but instead using a *Git* client with integrated *GitFlow* functionality, namely *GitKraken* [40].

4.2 Prototyping

4.2.1 Brainstorming

Brainstorming [12] is a formalized ideation process carried out within focus groups[2] to generate a great number of ideas early on the life-cycle of any project. This process has several purposes such as propose solutions to solve problems or propose new ways of working within the project.

Although brainstorming is a good technique to generate a big number of ideas, it is important to keep in mind that many of these ideas are not strong enough to be taken any further by themselves, as they have no more foundation other than the one inherent to the person who proposed them,

[2]In the game industry, focus groups are formed by a group of people who gathers to generate ideas for a game [12].

Instead, these ideas work as the basis of more complex ideas that need to be developed further in posterior phases of the project so they can become relevant to its success.

During the first days of the thesis development, rounds of brainstorming were always being held by the master thesis authors to understand the possibilities it could offer.

4.2.2 Paper Prototype

A paper prototype [12] is a quick draft made by using two of the most simple, yet powerful tools found when prototyping: pen and paper.

Unlike prototyping with other types of media, such as digital, paper prototypes are faster to set up than creating digital projects on a computer.

Therefore, in a short amount of time, a materialization of a design is brought to life from the head of the designer right to the surface of a paper which everyone can comment, criticize and make suggestions on.

In the developed game, paper prototypes were drawn regarding the whole level design, narrative design, but also, game design. This way, the master thesis authors could put form to their thoughts to show each other the intentions behind each of them in a simple and illustrative way.

4.3 Play-First Game Design

There are several valid ways to approach developing games from a design perspective.

A group of narrative designers can come up with a great story the development team wants to make real. Maybe all the team wants to design a new game featuring a specific aesthetic they want players to feel.

Far from these two examples, the play-first methodology is found [41].

The idea behind putting play first is to come up with a new, interesting way of playing. When the development team is focusing on designing and developing the core of the new game, this novel way of playing, everything else becomes irrelevant at this early stage of the development. All the resources are then invested in the creation of the perfect gameplay built around the aforementioned new way of playing.

Only once this new way of playing has been completely designed and developed and it is perfect according to the team's expectations, creativity and ambition, it is time to take another step: design and develop all the other parts of the game. However, the team needs to remember that everything else needs to be designed and developed with the idea of supporting the new way of playing behind. This will include, but it is not limited to, the game's narrative design, level design and sound design.

This whole method is underlined by the function principle, in which how all the game looks like is determined by how does it work, and not the other way round.

Nowadays, one of the companies that makes the most out of this methodology is Nintendo [42]. They only do develop games once they find an interesting and unique new way of playing, making a new video game around it.

In the developed game, every time new aspects of the games were about to be designed, they were implemented within a sandbox environment first. Here, they would be tested and polished until they met a specific set of quality standards. Only after that, they would be included in the game.

4.4 A. Cooper Principles

About Face: The Essentials of Interaction Design by A. Cooper [43] is one of the most recognized books in the field of interaction design. Game design is not so distinct from interaction design in terms of concepts and methods that can be applied in its development process. Therefore, some of the principles stated by Cooper can be applied directly or with some variations when performing game and level design tasks. The following subsections cover the most relevant of these principles used throughout the thesis development.

4.4.1 Pliancy and Hinting

Cooper uses the term *pliant* to refer to objects or screen areas that react to input that the user can manipulate. A common example of this is a button that can be clicked. Cooper stresses the importance of visually communicating pliancy, or equivalently, to provide pliancy hinting. The idea he defends is that those objects or screen areas that react to input, should visually communicate that they do so and how they do so. This communication can be performed in four different ways:

- **Static hinting**: When the object's pliancy is communicated by the static rendering of the object itself.
- **Dynamic hinting**: When the cursor passes over a pliant object, the object temporarily changes its appearance.
- **Pliant response hinting**: When the mouse is clicked but not released, the object shows that it is poised to undergo a state change.
- **Cursor hinting**: When the cursor's appearance changes as it passes over the object.

This principle greatly influenced the level design process. Authors applied it by making the level's elements visually communicate their purpose and whether they are interactive. Furthermore, a more direct application of this principle was carried out when designing the menus of the game.

4.4.2 Modeless Feedback

In the context of interaction and game design, feedback is referred to as the information given by the system to the user in relation to its state. *Modeless* feedback is a sub-type of feedback characterized by being built-in into the structure of the interface, not interrupting the flow of activities when presented.

This principle influenced both the game and level design process. In particular, authors designed the game so that feedback is presented organically by the level elements themselves, instead of through an HUD[3] interface layered on top.

4.4.3 Internal Coherence

An interactive product is said to possess internal coherence if all its parts behave consistently, providing the feeling of a unified whole. Designers must strive for coherence both in terms of visual appearance as well as in the means of interaction and its consequences. This means that not only the product should have a consistent visual identity, but also its components should react to the user in a consistent, predictable way.

This principle influenced the graphics and level design of the resulting video game. In particular, the creation of a strong visual identity that could be displayed throughout the whole game with no inconsistencies was one of the greatest challenges of the master thesis. The elements placed throughout the levels, when interactive, always look the same and react in the same way to the user's input. Nonetheless, the achievement of this consistency was not a trivial task. Ultimately, a framework and a formal method to ensure this consistency became a primal need.

[3]Heads-up display

5 Planning

5.1 Initial Planning

At the beginning of the master thesis, a planning report was written to set the initial scope of the project and elaborate an initial plan on how to develop it in the time given. However, as the development process progressed, the initial plan suffered many changes.

This section describes how was the initial planning as described in the aforementioned planning report.

5.1.1 Planned Result

In this subsection, a brief overview of the results expected to be obtained by the end of the master thesis is given.

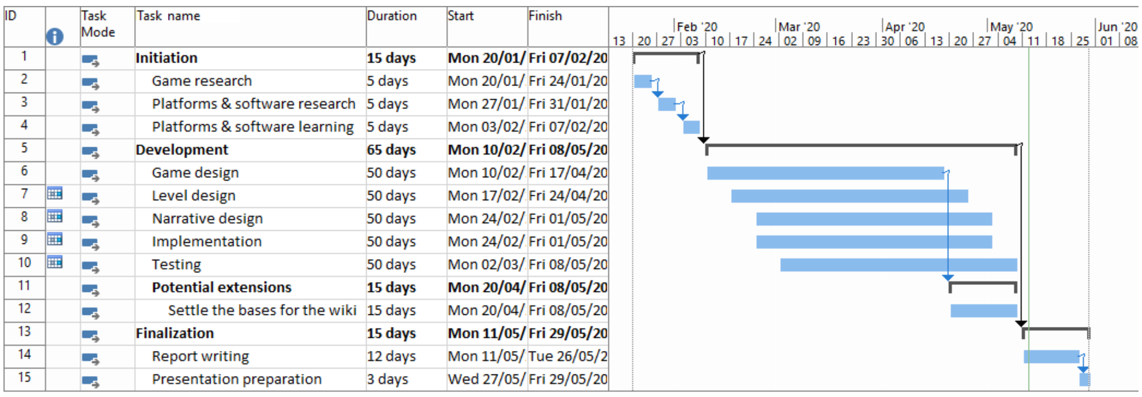
These are the expected results sorted by relevance and preference:

- Develop, or at least settle, the bases for a set of guidelines or an informative wiki providing specific information on how to design and implement physics-based interactions, as well as how these interactions affect to and behave with their surroundings.
- Develop a physics-based video game:
  - which helps to initiate an approach to the research problem described in Sec. 1.1, aiming to provide an answer to the research question set out in Sec. 1.2 via the scientific method.
  - in which the master thesis authors can break the bounds of their creativity, applying all the knowledge acquired during the whole Game Design & Technology Master's Programme and honing their game development skills while learning new of them.
  - which helps to the master thesis authors to increase the extension and quality of their professional portfolio. Therefore, it contributes to boost their chances to get into the game industry or continue their education following doctoral studies once they have completed the master's programme.

5.1.2 Time Plan

In order to elaborate an initial time plan supporting the planning of the master thesis, a Gantt chart[1] was created [44].

The Gantt chart in Fig. 5.1 represents the initial time plan for the master thesis. Note that the chart was intentionally not detailed, since the scope of the video game was not defined at that moment and, as such, it was impossible to predict which low-level tasks would be necessary by that stage of the development. Also, note that all the dates of the tasks were just early approximations as well.



ID		Task Mode	Task name	Duration	Start	Finish
1			**Initiation**	**15 days**	**Mon 20/01/**	**Fri 07/02/20**
2			Game research	5 days	Mon 20/01/	Fri 24/01/20
3			Platforms & software research	5 days	Mon 27/01/	Fri 31/01/20
4			Platforms & software learning	5 days	Mon 03/02/	Fri 07/02/20
5			**Development**	**65 days**	**Mon 10/02/**	**Fri 08/05/20**
6			Game design	50 days	Mon 10/02/	Fri 17/04/20
7			Level design	50 days	Mon 17/02/	Fri 24/04/20
8			Narrative design	50 days	Mon 24/02/	Fri 01/05/20
9			Implementation	50 days	Mon 24/02/	Fri 01/05/20
10			Testing	50 days	Mon 02/03/	Fri 08/05/20
11			**Potential extensions**	**15 days**	**Mon 20/04/**	**Fri 08/05/20**
12			Settle the bases for the wiki	15 days	Mon 20/04/	Fri 08/05/20
13			**Finalization**	**15 days**	**Mon 11/05/**	**Fri 29/05/20**
14			Report writing	12 days	Mon 11/05/	Tue 26/05/2
15			Presentation preparation	3 days	Wed 27/05/	Fri 29/05/20

Figure 5.1: Initial Gantt chart.

The process was divided into three stages: initiation, development and finalization.

5.1.2.1 Initiation

In the initiation stage, which lasts for three weeks, there is no development of the final product per se. Instead, research is the primary focus. In particular, research of relevant games, and also research related to the software and platforms that are used, such as game engines.

5.1.2.2 Development

In the development stage, the development work is performed. This stage is the longest one, starting right after the initiation stage is finished, and finishing approximately three weeks before the project ends. In this stage, the tasks that are tackled are game design, level design, narrative design, implementation and testing. As can be seen, most of the time these five tasks are performed in parallel, since they provide constant feedback to each other.

Furthermore, this stage includes one section called potential extensions. As the name of the section implies, it was created to contain tasks only tackled if there is enough time to do so, such as settling the bases for the wiki.

The task settling the bases for the wiki means to define its structure and introduce a list of elements it should contain, without elaborating each of them individually.

[1]A Gantt chart illustrates activities displayed against time.

5.1.2.3 Finalization

In the finalization stage, the report and the presentation are prepared. In these final weeks, writing the report is the primary focus of the team. After it is finished, the presentation is elaborated.

5.2 Final Planning

After approximately three weeks since the work started, the authors figured out that to make the most out of the project, some of the planned results needed to be changed. At this stage, this adjustment did not pose any major challenges, since the first three weeks were spent on research that was still relevant.

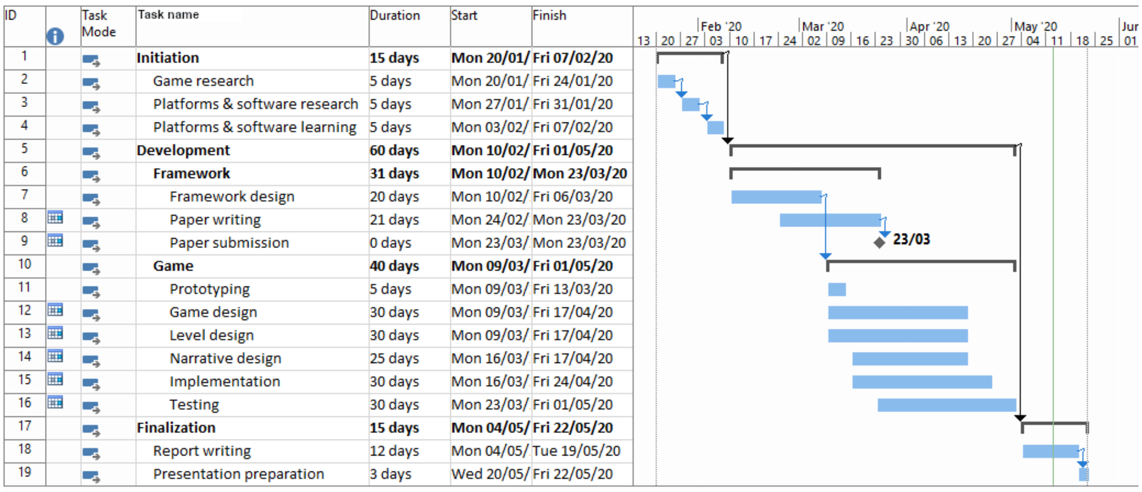
5.2.1 Planned Result

These are the expected results to obtain at the end of the thesis, sorted by relevance and preference:

- Develop a game design framework that aids the design and analysis of physics-based games. Submit this work to a relevant conference to obtain objective feedback from experts on the field and potentially have a published article.
- Develop a physics-based video game for the reasons explained in Sec. 5.1.1 and to apply the framework.

5.2.2 Time Plan

The Gantt chart in Fig. 5.2 represents the updated time plan for the thesis. This chart is not detailed since it represents a plan developed on an early stage, right after deciding to vary the planned result from Sec. 5.1.1. Consequently, the stated dates are just approximations. Small divergences are to be expected when carrying out each of the tasks.



ID	ℹ	Task Mode	Task name	Duration	Start	Finish
1		➔	**Initiation**	**15 days**	**Mon 20/01/**	**Fri 07/02/20**
2		➔	Game research	5 days	Mon 20/01/	Fri 24/01/20
3		➔	Platforms & software research	5 days	Mon 27/01/	Fri 31/01/20
4		➔	Platforms & software learning	5 days	Mon 03/02/	Fri 07/02/20
5		➔	**Development**	**60 days**	**Mon 10/02/**	**Fri 01/05/20**
6		➔	**Framework**	**31 days**	**Mon 10/02/**	**Mon 23/03/20**
7		➔	Framework design	20 days	Mon 10/02/	Fri 06/03/20
8	▦	➔	Paper writing	21 days	Mon 24/02/	Mon 23/03/20
9	▦	➔	Paper submission	0 days	Mon 23/03/	Mon 23/03/20
10		➔	**Game**	**40 days**	**Mon 09/03/**	**Fri 01/05/20**
11		➔	Prototyping	5 days	Mon 09/03/	Fri 13/03/20
12	▦	➔	Game design	30 days	Mon 09/03/	Fri 17/04/20
13	▦	➔	Level design	30 days	Mon 09/03/	Fri 17/04/20
14	▦	➔	Narrative design	25 days	Mon 16/03/	Fri 17/04/20
15	▦	➔	Implementation	30 days	Mon 16/03/	Fri 24/04/20
16	▦	➔	Testing	30 days	Mon 23/03/	Fri 01/05/20
17		➔	**Finalization**	**15 days**	**Mon 04/05/**	**Fri 22/05/20**
18		➔	Report writing	12 days	Mon 04/05/	Tue 19/05/20
19		➔	Presentation preparation	3 days	Wed 20/05/	Fri 22/05/20

Figure 5.2: Final Gantt chart.

Alike in Sec. 5.1.2, the process is divided into three main stages: initiation, development and finalization. Due to the change in the planned result, the development stage is significantly affected. However, the initiation and finalization stages remain as they were.

5.2.2.1 Initiation

Research in the fields of game design, philosophy and psychology is conducted. The outcome of the research is presented in Sec. 2, Sec. 3 and Sec. 4.

5.2.2.2 Development

As for the changes in the development stage, the most notable one is the creation of two sections: framework and game.

The framework section covers all the tasks related to the development of the formal process to design and analyze physics-based games, including the paper writing and its submission to a suitable conference within the duration of the master thesis.

The game section covers all the tasks required to develop a game using the framework. A new task, *prototyping* has been included. Its purpose its to cover all the preliminary steps for the development of the game, such as paper prototyping.

It is relevant to point out that the game development does not start until the framework is designed. This is intentional since the framework is applied when developing the game. However, both the paper writing and game development will be carried out in parallel, since at this stage no major changes in the framework are to be expected.

Another difference with respect to the initial planning (Sec. 5.1), is the fact that the time invested in developing the game has been considerably reduced from 65 days to 40 days. This decision was taken to obtain time to develop the framework, consequently reducing the scope and importance of the game itself. In the new plan, the game becomes an artifact to showcase the practical application of the framework instead of the main outcome of the thesis.

To conclude, this development process is presented in great practical detail in Sec. 6 where the whole process is divided into 7 different phases:

- **Phase 1: Prototyping**: The first prototype of *Under Surveillance* is made.
- **Phase 2: The PEP Framework development**: The framework and its correspondent paper are designed, written and submitted.
- **Phase 3: Design of Under Surveillance**: The framework is applied to design the game.
- **Phase 4: First version of Under Surveillance**: The first version of the game is developed.
- **Phase 5: Second version of Under Surveillance**: The second version of the game is developed.

- **Phase 6: Final version of Under Surveillance**: The third and final version of the game is developed.
- **Phase 7: Analysis**: The framework is applied to analyze *Angry Birds* and the thesis game.

5.2.2.3 Finalization

All the results are collected from both the framework and the developed game in Sec. 7. Next, the limitations, challenges and ethical considerations are explored and discussed in Sec. 8. Lastly, the conclusion of the master thesis as well as further work are presented in Sec. 9.

To finalize, the master thesis report is written the final presentation is prepared and presented.

5.2.3 External Resources

To develop the thesis, many external resources not created by the authors were used. These can be classified as software tools and assets.

5.2.3.1 Software Tools

The following tools have been used:

- **Adobe XD**: *Adobe XD* [45] is a user interface (UI) prototyping tool created by *Adobe Inc.*. This software was used to create high-fidelity prototypes of the menus of the video game. Among other similar alternatives, *Adobe XD* was chosen due to the familiarity authors had with this tool.
- **GitKraken**: *GitKraken* [40] is a *Git* [38] client created by *Axosoft.* It provides a user-friendly interface to operate the version control tool *Git*. *GitKraken* helped the authors to make the most out of the features of *Git* without having to learn its complex command-based interface. This tool was chosen because authors were familiar with it.
- **Microsoft Project**: *Microsoft Project* [46] is a tool developed by *Microsoft* that allows the creation of Gantt charts among other features. It was the chosen tool to create Gantt charts because authors were familiar with it.
- **Mixamo**: *Mixamo* [47] is a web service created by *Adobe Inc.* that provides downloads for rigged characters and animations. This tool was used because it allows to download any of the animations they provide, baked for any custom 3D modeled rigged character that the user uploads. This means that the authors can apply all the animations provided by this service to all the characters they included in the game.
- **Overleaf**: *Overleaf* [48] is a web application that allows collaborative creation and edition of *LaTeX* documents. Since both authors are involved in writing all the documents required for the thesis, using a collaborative tool was required. Moreover, the authors decided that *LaTeX* was the most appropriate

underlying software to write these documents, because of its widespread usage in the scientific community. *Overleaf* was consequently chosen not only because it meets all these conditions, but also because authors were familiar with its usage.

- **Unity**: *Unity* [49] is a cross-platform game engine developed by *Unity Technologies.* In order to meet the deadlines of the thesis while keeping the quality of the delivered work up to the authors' standards, it was decided to use a game engine to support the development of the game. After exploring alternatives such as *Unreal Engine* [50] or *Godot* [51], *Unity* became the chosen tool. The reason behind this decision is that it was determined that *Unity* provides all the functionality required to develop the envisioned game, while also being the game engine the authors were more familiar with.

5.2.3.2 Assets

Since the authors' skill set does not include all the relevant areas from game development, many external assets needed to be acquired. The assets, obtained via the *Unity Asset Store*, are:

- **Aura - Volumetric Lighting**: It simulates the scattering of the light in the environment and the illumination of micro-particles that are present in it but invisible to the eye or camera. This effect is also called volumetric fog [52].
- **Polygon Arsenal**: A low-poly bundle of roughly 550 particle systems [53].
- **Polygon - Sci-Fi City Pack**: A low-poly asset pack of characters, props, weapons, vehicles and environment assets to create Sci-Fi themed polygonal style games [54].
- **Polygon - Sci-Fi Space Pack**: A low-poly asset pack of space ships, characters, props, weapons, sound effects and environment assets to create Sci-Fi themed polygonal style games [55].
- **Retro Future Music Pack**: It contains 20 high-quality music tracks including loop versions of each track, suitable for retrowave, synthwave and cyberpunk atmospheres [56].
- **Sci-Fi Sound Pack**: 720 high-quality sound effects for Sci-Fi projects [57].
- **Water 2D Tool**: A tool conferring the ability to animate water [58].

5.2.3.3 Other Resources

Besides all the aforementioned resources, authors also employed other small, royalty-free assets they obtained. This was the case of the *Good Times* [59] font family, which is used in the user interface of the game. Furthermore, when the authors needed to have an icon, the *FlatIcons* [60] digital library was used since it provides royalty-free icons.

6

Execution

6.1 Phase 1: Prototyping

6.1.1 Game Design

At the beginning of the execution process, the first sessions of brainstorming were set up to find a new interesting way of playing worth developing for. As part of these sessions, several discussions were held but only one stood among them all.

In video games, players always need to perform interactions with the game world or the environment to progress further. They often perform many actions such as opening doors, shooting enemies or carrying objects around. In fact, performing these interactions and observing their outcomes is what makes video games special compared to other kinds of media. Nevertheless, rare is the case in which players can interact with every single element they might find in the environment. Usually, only physics-based games let the player interact with all the game world.

Based on these reflections, it was decided to develop a physics-based video game in which players would be able to interact with every single element placed in the game world and getting appropriate feedback from such interactions.

Therefore, the idea of letting the player cast and manipulate several natural elements, such as fire and water, was born in the form of the paper prototype found in Fig. 6.1.

In Fig. 6.1, in the first column, it is found the natural element the player would cast. In the second column, the environment element in which the spell would be thrown. From the third column onward, the result between the interaction of the natural element casting and the environment on which it is cast is found. For example, if fire is thrown onto ice, water would be produced. If more fire is thrown onto the resultant water, wind or smoke would be produced until there is no trace from the aforementioned ice.

Furthermore, as can be observed in Fig. 6.2, natural elements would be divided into two different categories: spells the player could cast and natural elements restricted only to the environment.

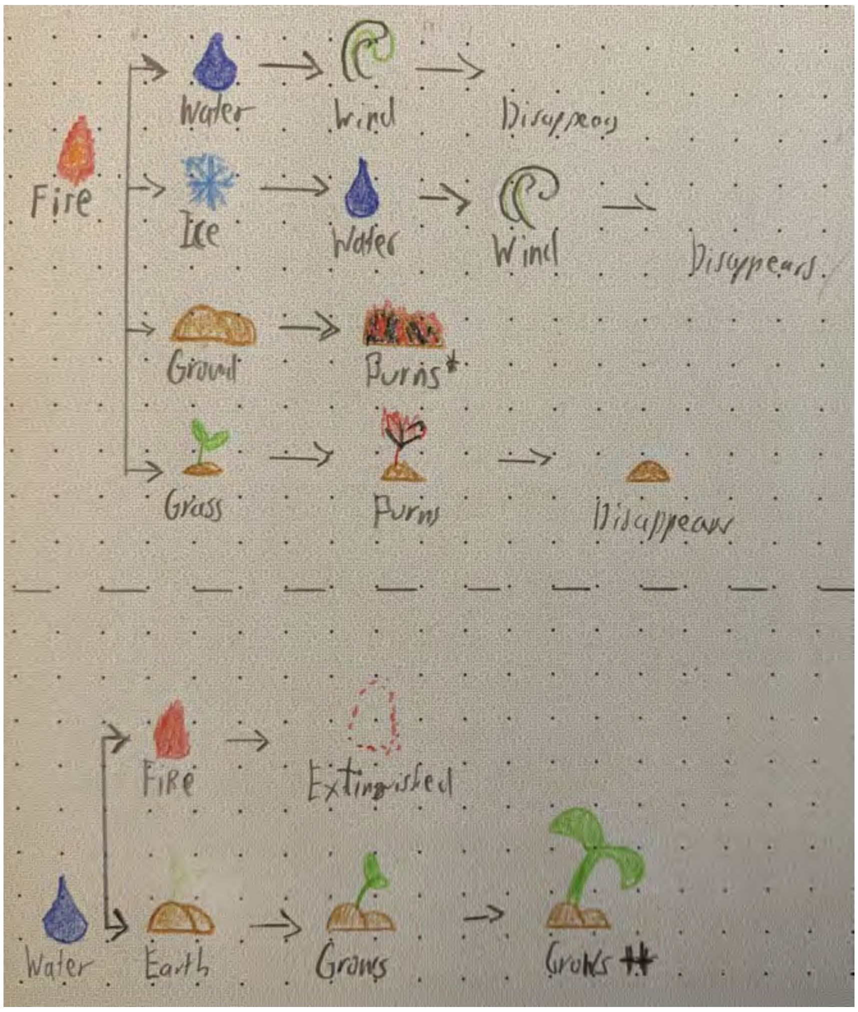
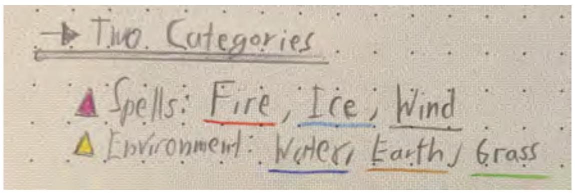


Figure 6.1: Paper prototype of the natural elements and the interactions between them.



→ Two Categories

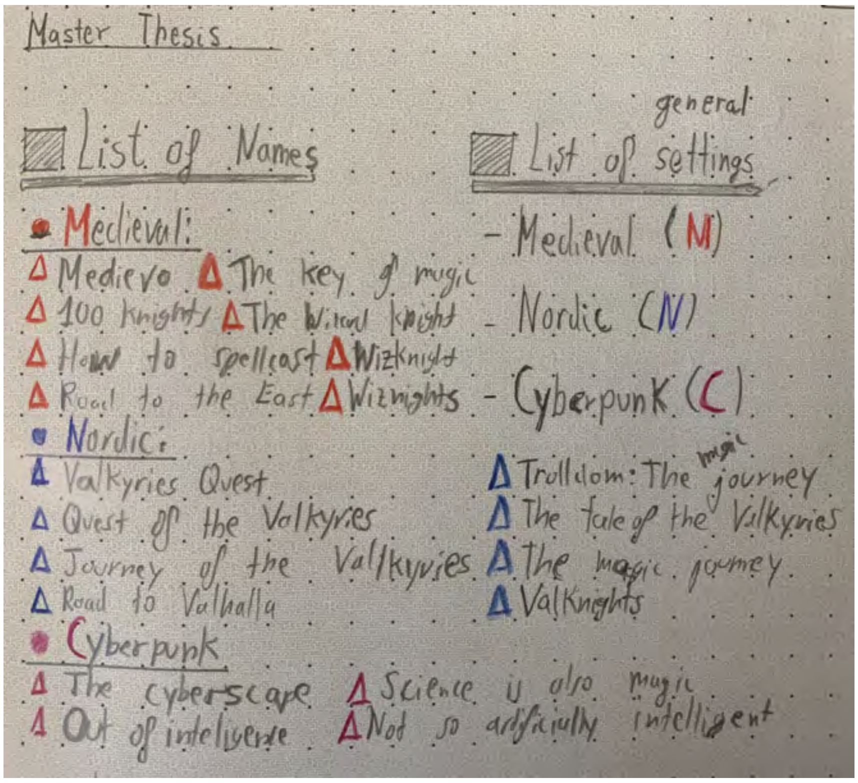
- Spells: Fire, Ice, Wind
- Environment: Water, Earth, Grass

Figure 6.2: Paper prototype of the categories of the natural elements.

6.1.2 Narrative Design

Once the core of the upcoming video game was prototyped as described in the previous section, a narrative was then built upon that.

After a new brainstorming session, many possible settings to host the aforementioned design were proposed but only three thematics were pushed forward: Medieval, Nordic, and Cyberpunk. Together with these three settings, a list of early possible names was also given as it is shown in Fig. 6.3. However, none of these names would make it to the end.



Master Thesis

▨ List of Names

● Medieval:

Δ Medievo Δ The key of magic
Δ 100 knights Δ The Wizard knight
Δ How to spellcast Δ Wizknight
Δ Road to the East Δ Wizknights

● Nordic:

Δ Valkyries Quest
Δ Quest of the Valkyries
Δ Journey of the Valkyries
Δ Road to Valhalla

Δ Trolldom: The magic journey
Δ The tale of the Valkyries
Δ The magic journey
Δ ValKnights

● Cyberpunk

Δ The cyberscape Δ Science is also magic
Δ Out of inteligence Δ Not so artificially intelligent

▨ List of general settings

- Medieval (M)
- Nordic (N)
- Cyberpunk (C)

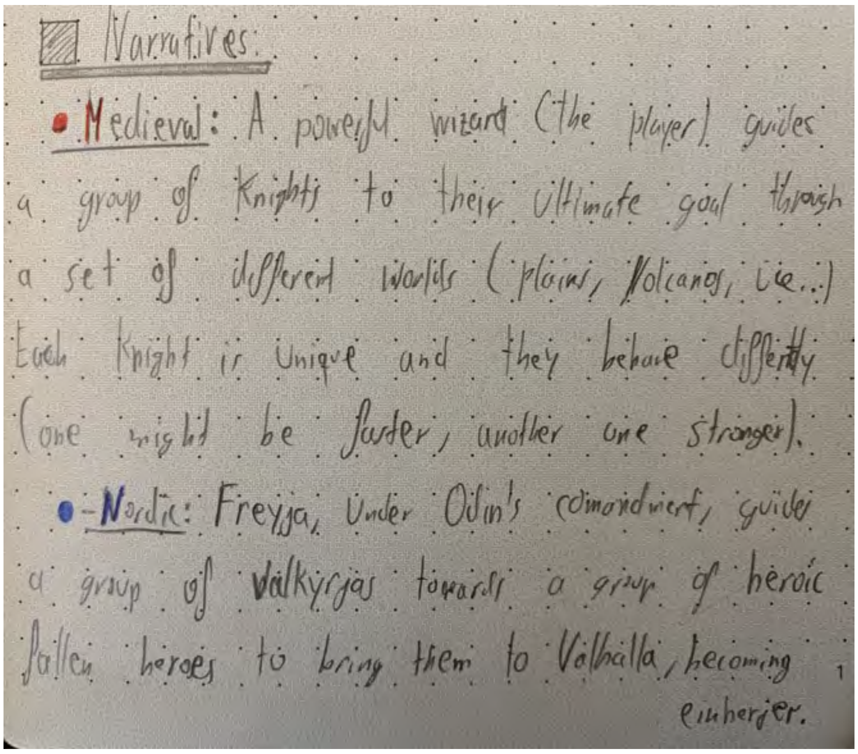
Figure 6.3: Settings and names to support the game design.

Right after this, the three thematics were developed a step further and three short narratives were written. They are presented in Fig. 6.4 and Fig. 6.5.

Together with the short narratives, the names the natural elements would have according to each narrative were listed in Fig. 6.6.

After careful consideration by the master thesis authors, it was decided to discard the Medieval and Nordic narratives and develop more in detail the Cyberpunk one. The reason behind this choice was made according to the fact that the authors thought that there are quite more medieval and nordic video games than cyberpunk ones. Moreover, they found more appealing the idea of making a cyberpunk video game.

Once the discard was made, the Cyberpunk narrative was taken further, always keeping in mind the new way of playing and how the narrative should support it.

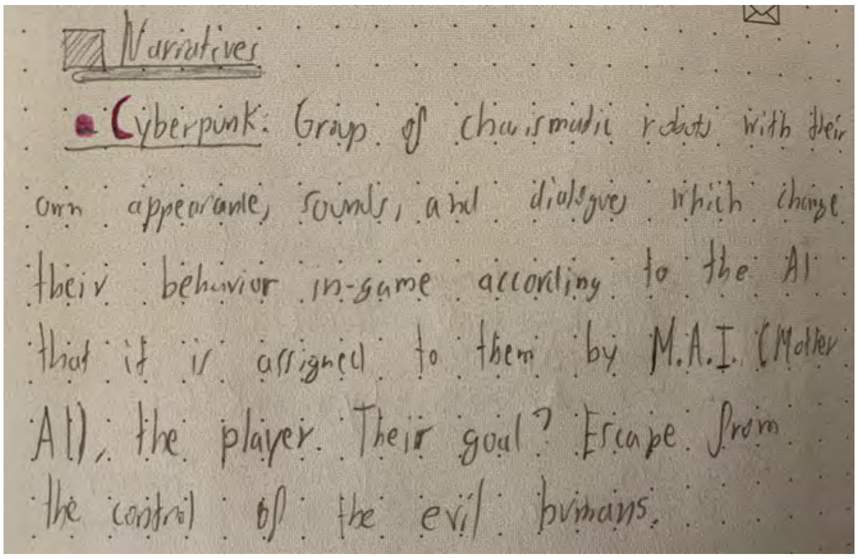


Narratives:

• Medieval: A powerful wizard (the player) guides a group of Knights to their ultimate goal through a set of different worlds (plains, volcanos, ice...) Each Knight is unique and they behave differently (one might be faster, another one stronger).

• -Nordic: Freyja, under Odin's commandment, guides a group of Valkyrjas towards a group of heroic fallen heroes to bring them to Valhalla, becoming einherjer.

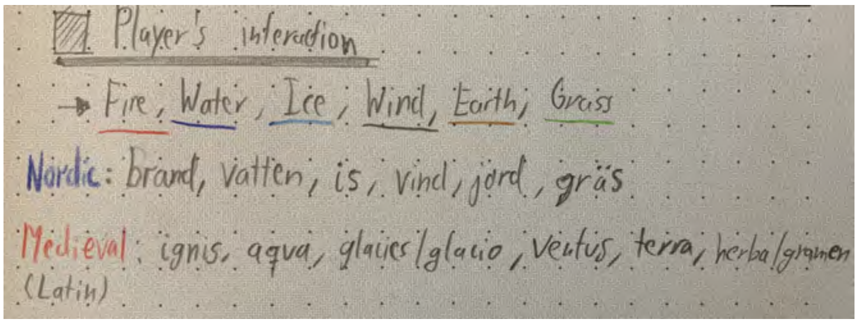
Figure 6.4: Medieval and Nordic short narratives.



Narratives

• Cyberpunk: Group of charismatic robots with their own appearance, sounds, and dialogues which change their behavior in-game according to the AI that it is assigned to them by M.A.I. (Mother AI), the player. Their goal? Escape from the control of the evil humans.

Figure 6.5: Cyberpunk short narrative.


▨ Player's interaction

→ Fire, Water, Ice, Wind, Earth, Grass

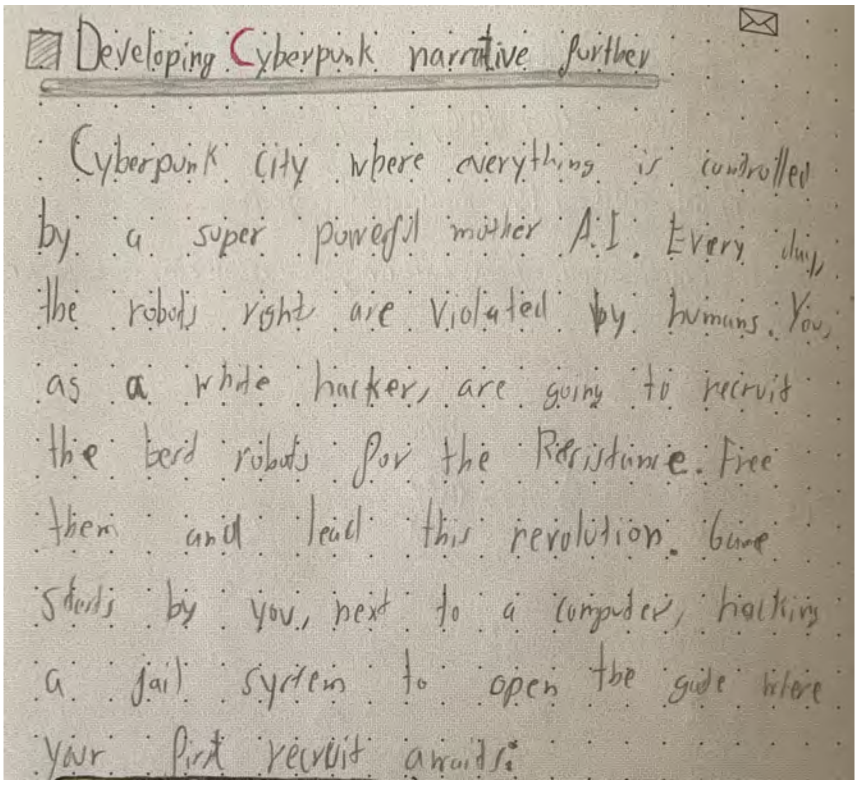
Nordic: brand, vatten, is, vind, jord, gräs

Medieval (Latin): ignis, aqua, glacies/glacio, ventus, terra, herba/gramen

Figure 6.6: Natural elements in Medieval, Nordic and Cyberpunk narratives.

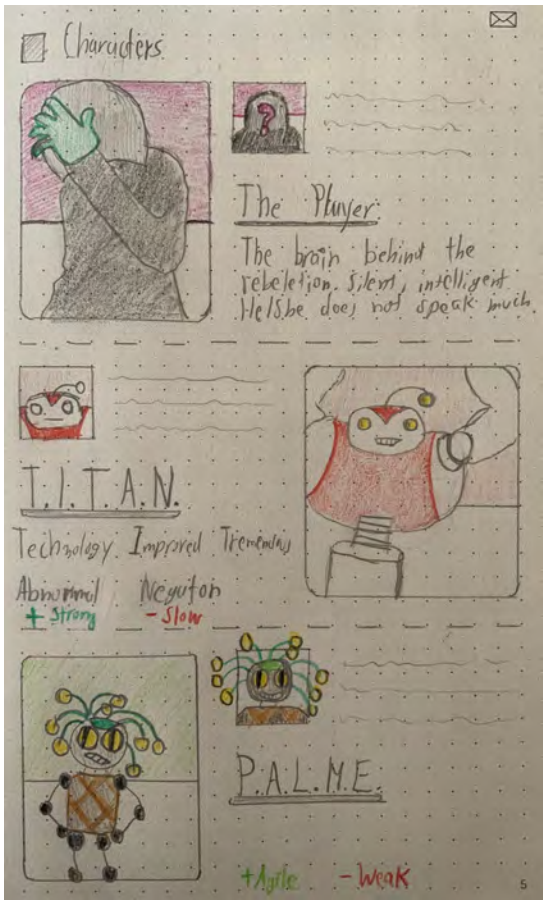
From this moment on, research on several games, books and movies, as described in Sec. 2.2, was performed before keep developing the Cyberpunk narrative. After the research was over, the name of *Under Surveillance* was assigned to the video game under development.

Fig. 6.7, Fig. 6.8 and Fig. 6.9 show how the narrative was evolving throughout several iterations. However, it was always kept in mind the design from Sec. 6.1.1 and how to support it through this narrative.

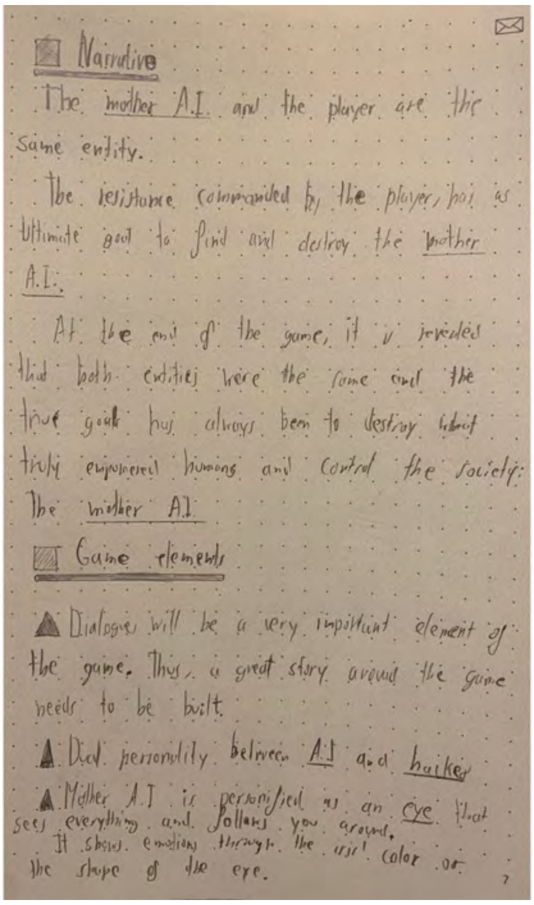

▨ Developing Cyberpunk narrative further

Cyberpunk city where everything is controlled by a super powerful mother A.I. Every day, the robots right are violated by humans. You, as a white hacker, are going to recruit the best robots for the Resistance. Free them and lead this revolution. Game starts by you, next to a computer, hacking a jail system to open the gate where your first recruit awaits.

Figure 6.7: Cyberpunk narrative developed further in detail.


Characters
The Player:
The brain behind the rebelletion. Silent, intelligent. He/She does not speak much.
T.I.T.A.N.
Technology Improved Tremendously Abnormal Neyuton
+ Strong
- Slow
P.A.L.M.E.
+Agile
- Weak
5

(a) Game characters.


Narrative

The mother A.I. and the player are the same entity.

The resistance commanded by the player, has as ultimate goal to find and destroy the mother A.I..

At the end of the game, it is revealed that both entities were the same and the true goal has always been to destroy what truly empowered humans and control the society: The mother A.I.

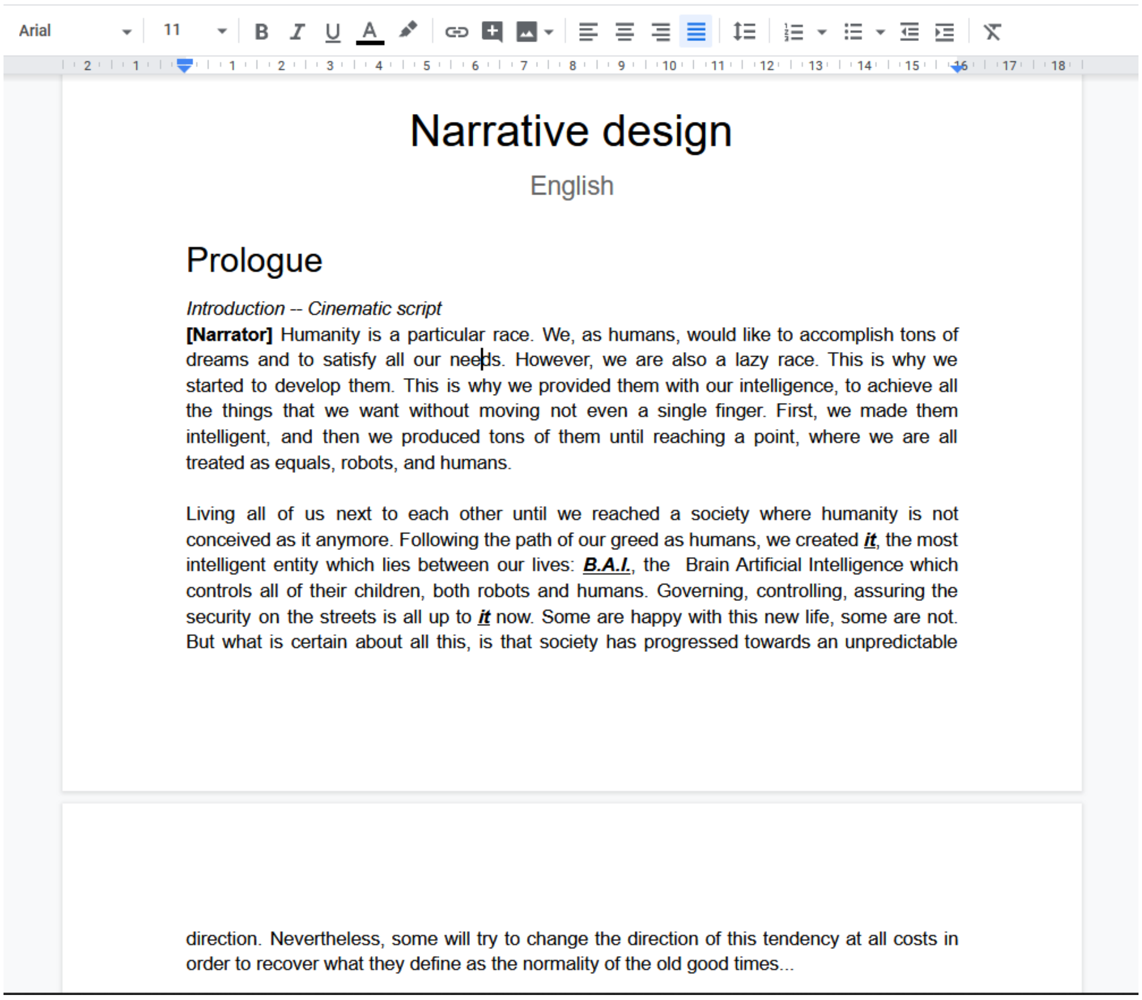
Game elements

- Dialogue will be a very important element of the game. Thus, a great story around the game needs to be built.
- Dual personality between A.I and hacker
- Mother A.I is personified as an eye that sees everything and follows you around. It shows emotions through the iris' color or the shape of the eye.

7

(b) Most relevant game narrative elements explained.

Figure 6.8: Game characters and most relevant game narrative elements explained.



Narrative design

English

Prologue

Introduction -- Cinematic script

[Narrator] Humanity is a particular race. We, as humans, would like to accomplish tons of dreams and to satisfy all our needs. However, we are also a lazy race. This is why we started to develop them. This is why we provided them with our intelligence, to achieve all the things that we want without moving not even a single finger. First, we made them intelligent, and then we produced tons of them until reaching a point, where we are all treated as equals, robots, and humans.

Living all of us next to each other until we reached a society where humanity is not conceived as it anymore. Following the path of our greed as humans, we created ***it***, the most intelligent entity which lies between our lives: ***B.A.I.***, the Brain Artificial Intelligence which controls all of their children, both robots and humans. Governing, controlling, assuring the security on the streets is all up to ***it*** now. Some are happy with this new life, some are not. But what is certain about all this, is that society has progressed towards an unpredictable direction. Nevertheless, some will try to change the direction of this tendency at all costs in order to recover what they define as the normality of the old good times...

Figure 6.9: Cinematic prologue script.

6.1.3 Level Design

Once the narrative design of *Under Surveillance* was defined, the next step was to initiate the level design.

Subsequently, developing *Under Surveillance* as a 2.5D physics-based puzzle adventure video game was decided. It was chosen to make the game in 2.5D as it is easier to develop than a fully 3D game. Furthermore, puzzle-solving video games rely more than the usual on players' interactions, consequently supporting the design described in Sec. 6.1.1.

Therefore, a couple of paper prototypes regarding how the first level of *Under Surveillance* should be were made. This is illustrated in Fig. 6.10 and Fig. 6.11.

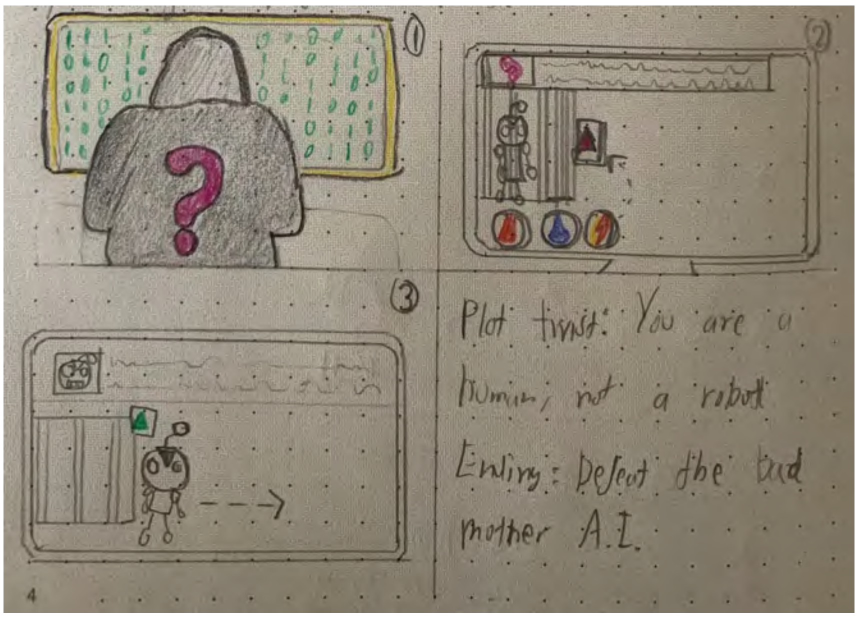

①
②
③
Plot twist: You are a human, not a robot
Ending: Defeat the bad mother A.I.
4

Figure 6.10: The player takes control of the game and helps the robot to escape from the jail they are confided to by throwing electricity to the control panel of the cell.

6.1.4 Post-Mortem

Once the phase of prototyping was over, the master thesis authors realized that the game design was far from perfect. Despite, the importance of the interactions between the player and the environment and the resultant effects from those, they were never taken into account in the design as they should have. This was a big challenge yet to be confronted to answer the research question from Sec. 1.2.

At this point it was decided to freeze the development of *Under Surveillance* and continue with the development of a formal process or framework aiming to ease the task of designing a video game of these characteristics. The drafts from Fig. 6.12 were the first steps made in the urge of this need, which eventually became the PEP Framework as defined in the next section.

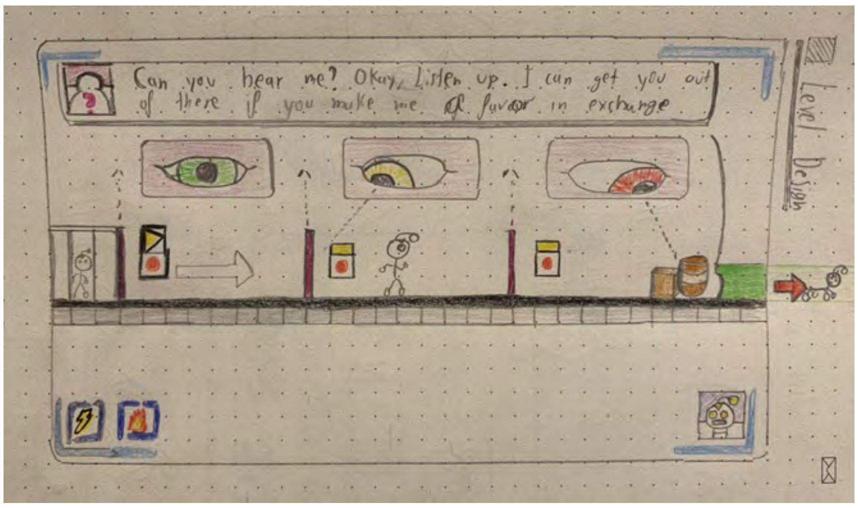
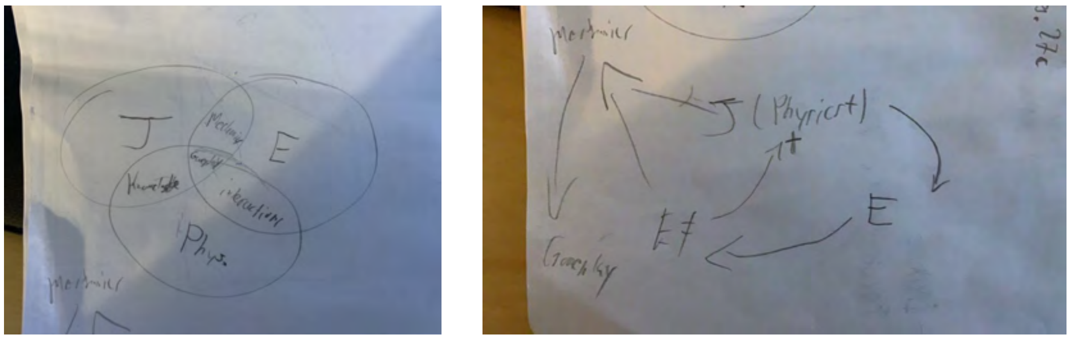

Can you hear me? Okay, Listen up. I can get you out of there if you make me a favor in exchange
Level Design

Figure 6.11: The player helps the robot to escape from the jail they are confined to.



(a) Main components. (b) Secondary components.

Figure 6.12: Drafts regarding the main and secondary components of the PEP framework.

6.2 Phase 2: The PEP Framework development

The PEP framework (standing for *Player*, *Environment* and *Physics)* is a formal method that guides the design and analysis of physics-based games[1]. It ensures consistency while making intuitive to the player the emerging behaviors that are the consequence of these game's physics.

According to the PEP framework, physics-based games can be broken into three main components and three secondary components born from the interactions between the former. In Fig. 6.13 it is illustrated how all these components relate to each other.

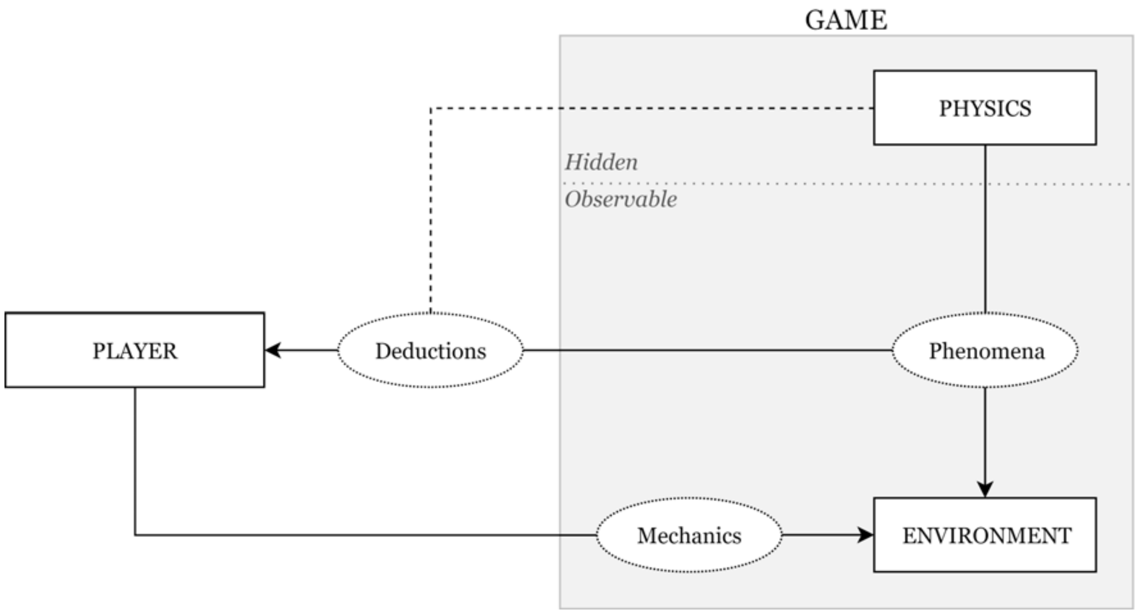


Figure 6.13: The PEP framework architecture.

The three main components are:

- **Physics**: Set of rules governing the *environment*.
- **Environment**: Materialization of the current game state [10].
- **Player**: External user who interacts with the *environment*.

The three secondary components are:

- **Mechanics**: Methods invoked by the *player*, designed for interaction with the *environment* [61].
- **Phenomena**: Observable events or occurrences manifested in the *environment*, consequence of the *physics*.
- **Deductions**: *Player*'s knowledge gain regarding how the *physics* work derived from observed *phenomena*.

[1]This includes, but it is not restricted to, board games and video games, i.e., games featuring real physics or simulated physics.

6.2.1 The PEP Framework in Detail

Along this section, it is described each of the PEP framework main and secondary components in further detail. At the same time, explicit relations between these concepts and the ones previously described in Sec. 2.1 are established.

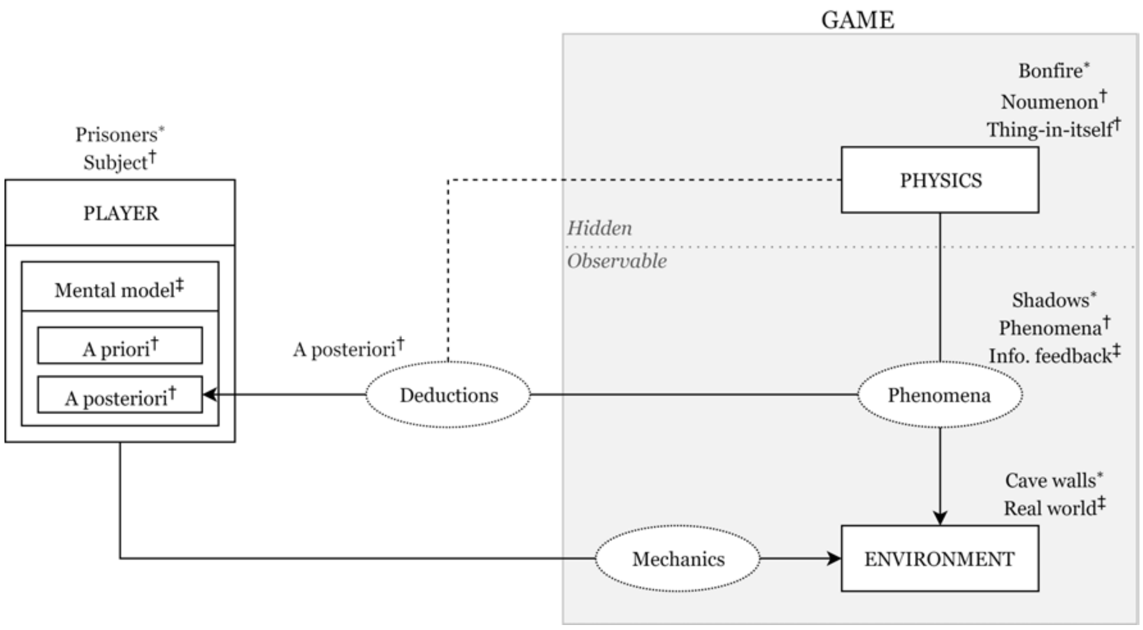


Figure 6.14: The PEP framework architecture in detail. Related Plato's (*), Kant's (†) and mental model and double-loop learning (‡) concepts are shown.

6.2.1.1 Physics

Physics are the set of rules governing the *environment* of physics-based games. This includes, but it is not limited to, Newton's laws of motion, thermodynamics' laws and the electromagnetic force.

Nevertheless, there are video games featuring *physics* not present in the real world, leading to unique behaviors existent only in these games. A clear example of this is found in *The Legend of Zelda: Breath of the Wild* [62] and Link's Stasis Rune. Link can use it to temporarily freeze objects in time, storing potential energy while the effect lasts. When the effect is over, all the corresponding kinetic energy is then liberated.

In relation to the *Allegory of the Cave*, the bonfire is equivalent to the *physics*. Given the fact that prisoners cannot see the bonfire, the interpretations they make regarding how the shadows are originated are potentially wrong. Likewise, the *physics* remain hidden to the *player* who can only interpret how they work by observing the *environment* and the emerging *phenomena*. The *physics* are the reason why *phenomena* emerge in the *environment* like the bonfire is the reason behind how the shadows are projected on the wall.

In Kant's philosophy, the noumenon, or thing-in-itself, is equivalent to the *physics*. The noumenon cannot be perceived by the subject as the *physics* cannot be directly observed by the *player*. In despite of this, the *physics* govern the *environment*, hence they are the noumenon which instantiates the *phenomena*.

6.2.1.2 Environment

The *environment* is the materialization of the current game state. It is considered as the whole game world including all the game objects, characters and even the *player*'s avatar[2]. Given the physics-based video game, *Angry Birds* [63] as an example, later renamed as *Angry Birds Classic*, the *environment* includes all birds, pigs, the slingshot, wooden planks, glass blocks, etc.

In the *Allegory of the Cave*, the walls correspond to the *environment*. In these observable walls, the shadows remain visible to all the prisoners. Analogously, in the PEP framework, the *environment* hosting emerging *phenomena* is observable by the player during gameplay.

Furthermore, the real world of the double-loop learning model is equivalent to the *environment* as they are both observable elements in which feedback and *phenomena* emerge, respectively.

6.2.1.3 Player

The *player* is an external user to the game who interacts with it via the use of game *mechanics*.

In the *Allegory of the Cave*, the prisoners and the *player* are correlative. The prisoners can only perceive the projected shadows while the bonfire and the people carrying the objects are not visible to them. Correspondingly, the *player* can only perceive the emergent *phenomena* in the *environment* while the *physics* remain hidden to them.

In relation to Kant's philosophy, the subject is equivalent to the *player*, who perceives *phenomena* and gathers empirical evidence. From this evidence, a posteriori knowledge, in form of *deductions* on how the *physics* work, is built.

The *player*'s a posteriori knowledge modifies and improves their mental model of the *physics* and how they work. This mental model is supported by the *player*'s a priori knowledge too. From a Kantian perspective, a priori knowledge is not derived from any experience in the real world. Similarly, from the PEP framework perspective, the *player*'s a priori knowledge is not derived from any experience within the game, but it can be derived from any external experiences to the game itself, such as the ones lived in the real world or while playing other games. How gravity works in the real world is an example of *player*'s a priori knowledge. The *player* makes good use of this knowledge to shape an early mental model of how gravity might work in the game. Eventually, the gameplay makes the *player* to compare how gravity works both in the real world and in the game, leading to a posteriori knowledge and a more accurate mental model of how it behaves in the latter.

[2] *Player*'s representation in the *environment*.

6.2.1.4 Mechanics

The *mechanics* are methods invoked by the *player* to interact with the game. These interactions can only be carried out within the *environment*. Moreover, the *environment* and, equivalently, the game state may change by performing *mechanics*.

6.2.1.5 Phenomena

The *phenomena* are observable events or occurrences which manifest in the *environment* as a direct consequence of the game's *physics*. These visible manifestations are perceived by the *player*.

In physics-based games, falling objects due to gravity, burning objects due to fire and the laws of thermodynamics and the propagation of electricity in conductive materials due to the fundamental laws of electricity are common examples of *phenomena*. Only the designers' creativity set the limits of the different kinds of *phenomena* in this type of games.

As it has already been stated, the *physics* remain hidden to the player while the *phenomena* is the only visible manifestation of them. Therefore, the *player* can only understand to some extent how the *physics* work by carefully observing these *phenomena* in the game.

Regarding the *Allegory of the Cave*, the shadows projected on the wall are equivalent to the *phenomena*. These shadows are a visible manifestation consequence of the bonfire in conjunction with the group of people carrying objects around. When the prisoners perceive these shadows, the only observable event they can interpret, they try to understand the rules of the world they inhabit. Likewise, the *physics* governing the *environment* can only be understood via the *phenomena* players observe.

Concerning Kant's philosophy, the definition Kant gives about *phenomena* is analogous to how it is defined in the PEP framework. Kant's *phenomena* emerge from the noumenon or thing-in-itself constituting observable events in the same way the PEP framework's *phenomena* emerge as a consequence of the *physics*.

According to what the information feedback means in double-loop learning, a strong connection between such information and the *phenomena* is present. Information feedback is obtained when facing goals and challenges within the real world, letting people learn from their ways to approach them and their results, hence updating their mental models. The same procedure when the *player* learns about their surrounding *environment* via *phenomena* is given and the *player*'s mental model of the *physics* is improved.

6.2.1.6 Deductions

Deductions are the *player*'s knowledge gain regarding how the *physics* work according to the observed *phenomena*. All this knowledge helps the *player* to form a mental model on how the *physics* might work and which *phenomena* might emerge from applying certain *mechanics* to the *environment*. However, the *player* will never have the certainty of how close their mental model got to the actual game's *physics*, although a very close approximation is to be expected if the game is consistent and intuitive to the *player* at all times.

As for Kant's philosophy, a posteriori knowledge is considered as a set of *deductions* by the PEP framework. According to all the *phenomena*, players observe and the empirical evidence they gather, their *physics*' mental model is updated. Alike a posteriori knowledge, *deductions* are not universal. *Deductions* may explain the observed *phenomena* but they are not enough to fully understand the underlying nature of the *physics*.

To conclude, acquiring knowledge is an iterative process in which the *player* gets a better understanding of the *physics* the more *phenomena* they observe and the more *deductions* they make from them.

6.2.2 Applying The PEP Framework

This section provides a list of steps to follow when applying the PEP framework from two different perspectives: design and analysis.

From the design point of view, the procedure starts by creating the *phenomena* and then deriving the *environment* from them. These first two steps ensure consistency between both of them, since every *phenomenon* is thereafter supported by at least one element from the *environment*. Next, the *physics* and game *mechanics* are created by taking into account the existing *phenomena* and *environment*. This guarantees a set of *physics* covering all the possibilities of the *phenomena* and *environment*, while at the same time every *mechanic* supports the emergence of *phenomena*. As the final step, the *deductions* a *player* might make during gameplay are designed to control how *players* learn about the game's *physics*.

From the analysis perspective, the method explains how to make use of the framework as a formal approach to analyze physics-based games to review their consistency. It also helps game designers to detect problems and avoid repeating the same mistakes. The analysis starts by identifying in no particular order the game *mechanics*, the *phenomena* and the *environment* elements. These elements are then validated to check whether they are consistent with each other. Finally, the *deductions* a *player* can make are evaluated to check if the *player* might be overwhelmed when learning how the game's *physics* work during gameplay.

6.2.2.1 Design

This subsection explains the formal process on how to use the PEP framework as part of the design and development of new physics-based games. This process consists of the following steps:

1. **Design phenomena**: In this creative task, game designers propose *phenomena* which will be present in their new physics-based game depending on what type of game they want to create.
2. **Design environment**: Taking as a starting point the *phenomena* designed in the previous step, an *environment* is derived from them. How designers

want the *environment* to be like has a considerable impact on how important the *player*'s a priori knowledge will be in the game. If the *phenomena* are typically found in the real world or other games played by the *player*, then the *player* will make use of a substantial amount of their a priori knowledge to understand how the *physics* work. On the other hand, if the *player* has rarely or never witnessed the *phenomena* emerging in this particular *environment*, the *player* will not be able to apply any a priori knowledge. Thus, the *player* will only rely on their a posteriori knowledge gained during gameplay through *deductions*.

3. **Formalize physics**: Once the *phenomena* and the *environment* have been defined, the *phenomena* behaviors in such *environment* are generalized by defining laws of *physics*. These set of laws will rule all the *environment* and all its emerging *phenomena*.

4. **Design mechanics**: This step is interchangeable with the previous one, since having the *phenomena* and the *environment* defined suffices to start designing game *mechanics*. In order to design the *mechanics*, how the *player* will interact with the *environment* must be specified. These *mechanics* must support the *phenomena* by allowing the *player* to directly or indirectly originate, alter or bring to an end *phenomena*.

5. **Design deductions**: All the previous steps aim to improve consistency in physics-based games by ensuring the *phenomena*, the *environment* and the *mechanics* exist for a justifiable reason. However, besides consistency, the framework also strives to take into account how the *player* can learn and understand how the *physics* work in these games. In this regard, in order to ensure a game in which its *physics* are intuitive to the *player*, the amount of *deductions* the *player* needs to make must be limited during gameplay. This can be achieved in two different ways. First, by designing more familiar *phenomena* and *environment* to the *player*, so more a priori knowledge is used. Secondly, by designing game levels where all *phenomena* and *environment* are gradually introduced, to avoid overwhelming situations to the *player* in terms of learning.

6.2.2.2 Analysis

This subsection describes the formal process on how to make use of the PEP framework as part of the analysis to break an existing physics-based game down. It consists of the following steps:

1. **Identify the phenomena, the environment and the mechanics**: All the *phenomena*, the *environment* elements and the *mechanics* are listed. Since the game is already designed, these can be identified in any particular order.

2. **Validate the phenomena and the environment**: The *phenomena* identified in the previous step must be validated by checking whether or not the *environment* supports such *phenomena*. In any state of the game, given the same *environment* elements, it must be checked if the same *phenomena* emerge.

If they do not, a lack of consistency is then detected.

3. **Validate the mechanics**: The *mechanics* identified in the first step must be validated by checking if they directly or indirectly support the *phenomena*. *Mechanics* which do not fulfill this condition, have no relevance to the gameplay, hence they have no positive impact on the game's consistency.

4. **Evaluate the deductions**: The previous steps examine the consistency of a physics-based game by checking whether its proper *phenomena* emerge, supported by its *mechanics*, according to its *environment*. Nevertheless, how the *player*'s cognitive process on understanding how the *physics* work in these games is also tackled in this analysis process. Therefore, the *player*'s *deductions* must be evaluated too. To perform this task, it must be determined first the a priori knowledge most of the *players* have available before playing the game for the first time. Then, it must be studied at which points of the game the *phenomena* emerge. Every *phenomenon* which cannot be explained by the *player*'s a priori knowledge will lead them to make *deductions*, establishing a posteriori knowledge about how the game's *physics* work. The more gradually the *player* needs to make *deductions*, the less overwhelming the game becomes when learning about it and its *physics*.

6.3 Phase 3: Design of Under Surveillance

To design the game that is part of the thesis, *Under Surveillance*, the PEP framework has been applied, following the formal process described in Sec. 6.2.2.1:

6.3.1 Design Phenomena

Right from the beginning, the master thesis authors envisioned a game featuring electricity, water and fire behaving as they do in the real world. Thus, the following *phenomena* were proposed:

- **Electricity**: Propagation through conductive materials and interaction with electrical devices.
- **Water**: Movement according to fluid dynamics, buoyancy of different objects, vaporization due to hot temperatures and freezing due to cold temperatures.
- **Fire**: Propagation through burnable materials, smoke generation and extinction in contact with water.

6.3.2 Design Environment

The master thesis authors envisioned a game in which the challenge was not to figure out how electricity, water and fire behave, but rather how to use these elements to solve the puzzles present along the way. Therefore, letting the *player* make use of their a priori knowledge as much as possible was a must. Consequently, elements typically found in the real world interacting with the aforementioned *phenomena* were designed as part of the *environment*. These *environment* elements, classified by whether they typically interact with electricity, water or fire are:

- **Electricity**: Metallic and non-metallic objects which conduct and isolate electricity, respectively, and electrical devices that can be turned on or off.
- **Water**: Shapes allowing the fluid dynamics *phenomena* to emerge, such as slopes or empty pools, and objects with different buoyancies which might float or sink when put in water.
- **Fire**: Objects made of flammable materials, such as wood, and nonflammable materials, such as steel.

6.3.3 Formalize Physics

According to the aforementioned designed *phenomena* and *environment*, the *physics* were generalized. Since the electricity, water and fire emulate their behaviors in the real world, simplifications of actual laws of *physics* were considered.

6.3.4 Design Mechanics

Given the need to support the *phenomena* with simple, yet powerful *mechanics*, the *player* is provided with the possibility to interact with the *environment* by throwing electricity, water or fire to any *environment* element.

6.3.5 Design Deductions

So far, the *phenomena* and the *environment* were designed so as much as possible a priori knowledge is used by the *player*. Moreover, the *player* is not able to use electricity, water and fire from the beginning of the game, but these are gradually introduced as they progress. These two design decisions sort and limit the number of *deductions* the *player* needs to make to gain a posteriori knowledge. Thus, the *player* can now intuitively learn and understand how the formalized *physics* work as the *deductions* they might make have been adjusted, avoiding overwhelming situations in terms of learning.

6.4 Phase 4: First Version of Under Surveillance

Having the game designed after applying the PEP framework in the previous section, the development of *Under Surveillance* was resumed. In this phase, an overview of the different aspects of the first version of the game is provided.

6.4.1 Character

The main character of *Under Surveillance* is a cyberpunk-stylized woman robot, included in the *Polygon - Sci-Fi City Pack* asset package, shown in Fig. 6.15. However, the player does not directly control this character. Instead, there is a virtual entity representing the player, *The Hacker*, that can interact with the environment to create new paths for the main character to advance automatically through them.

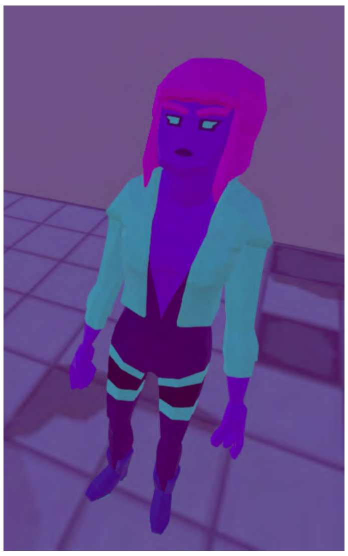


Figure 6.15: The main character of the first version of *Under Surveillance*.

To implement this behavior, *Unity*'s `NavMesh` [64] system was used. This system requires the developers to define an agent that will move in through the game world, as well as walkable areas. At runtime, the agent's destinations in terms of world-space coordinates are specified. The agent will then move automatically towards the specified position using a valid path according to the previously defined walkable areas. In Fig. 6.16 the `NavMesh` is illustrated.

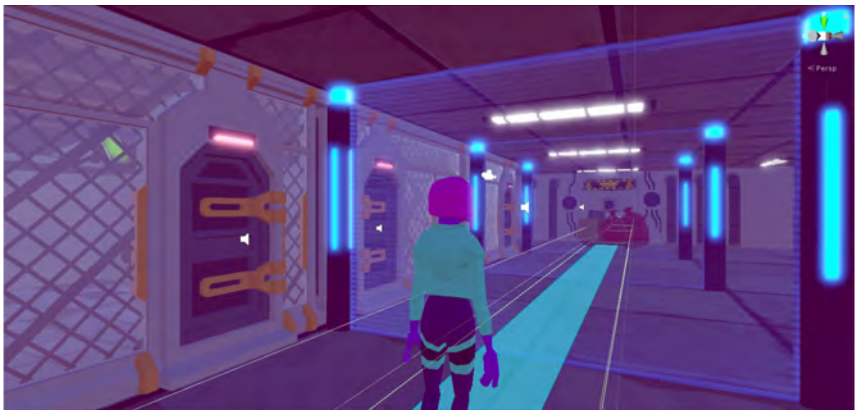
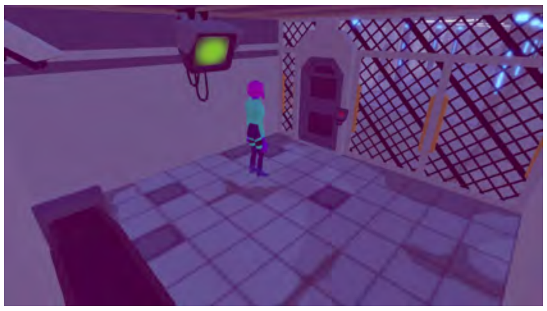


Figure 6.16: The `NavMesh` is represented as a blue carpet on the floor.

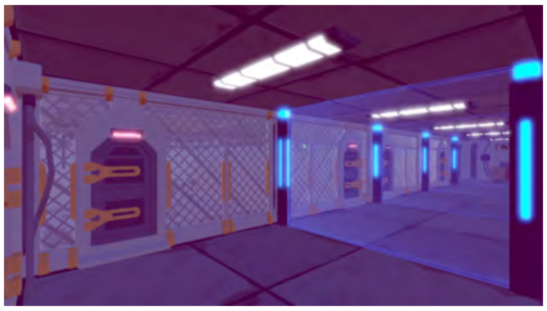
Concerning the physics-based elements of the game, namely electricity, water and fire, the idea was that the player could select these elements in the UI and then click anywhere in the game world to make *The Hacker* use them. However, this version ended up discarded before implementing these features.

6.4.2 Level Design

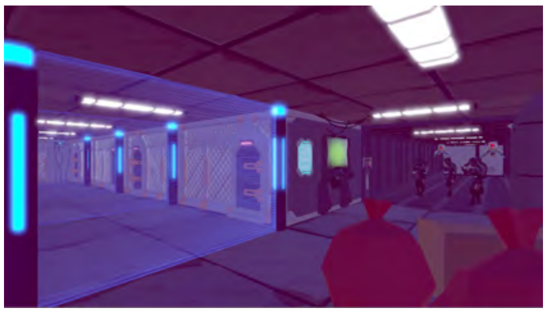
It is important to keep in mind that this first version of the game was discarded when it was still a prototype. Thus, most of the interactions with the environment were not implemented. This section will describe how the designed level was structured and what were the planned interactions. However, detailed aspects of the implementation are omitted. In Fig. 6.17, an overview of the whole level is provided.



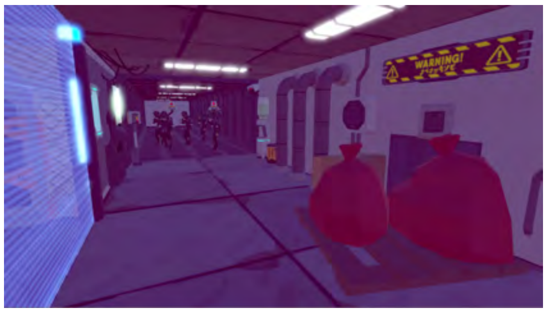
(a) The jail cell of the main character.



(b) Beginning of the corridor.



(c) Intersection of the corridor.



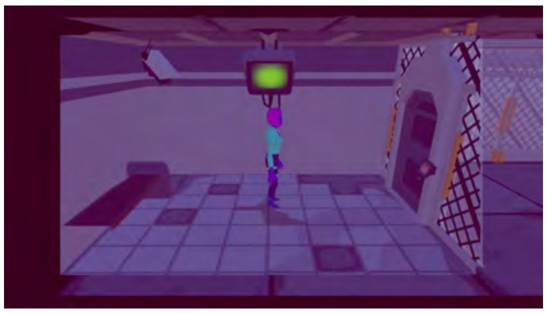
(d) Ventilation shaft hidden behind a pile of objects.

Figure 6.17: Level Design of the first version of *Under Surveillance* overview.

All the levels *Under Surveillance* and their contents were entirely built by making use of all the assets included in *Polygon - Sci-Fi City Pack* and *Polygon - Sci-Fi Space Pack*.

The game starts with the main character inside of a jail cell. The player is supposed to help her escape by using electricity to hack the door of the jail cell. After this, the character automatically moves to the right, leaving the jail cell behind and arriving at one long corridor. This sequence is shown in Fig. 6.18.

Once in the corridor, the player can observe other prisoners that are also locked in jail cells. This scenery aims to provide hints on the narrative of the game. The prisoners are animated, using animations from *Mixamo*, displaying them with different moods. Furthermore, it is possible to observe throughout the whole level several cameras located on the walls. These cameras are always pointing towards the player's position, conveying the feeling that the player is being observed at all times. In Fig. 6.19, a prisoner and a camera from a jail cell are illustrated.


(a) The main character imprisoned in the jail cell.

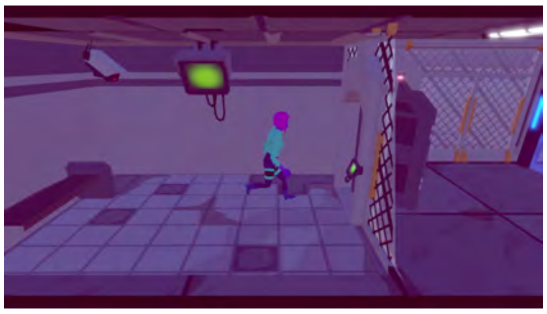

(b) The main character leaving the jail cell.

Figure 6.18: Jail cell area.

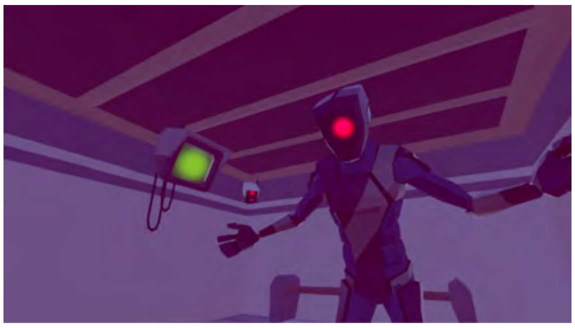
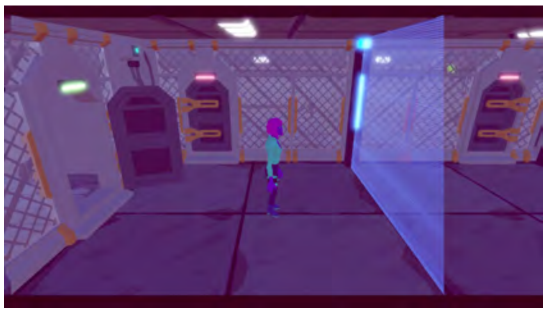


Figure 6.19: Prisoner observed by a camera inside of a jail cell.

In order to proceed further in the level, the player must use electricity to disable some electric barriers that are blocking the way. Once these barriers are disabled, the main character reaches an intersection point. This sequence is shown in Fig. 6.20.


(a) Beginning of the corridor.

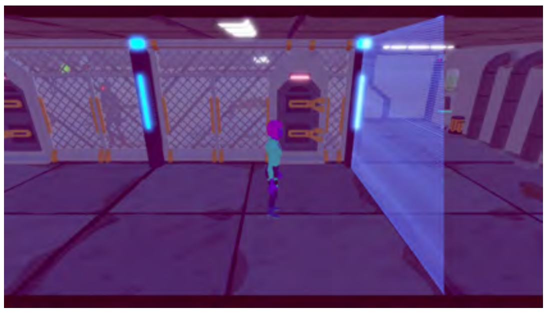
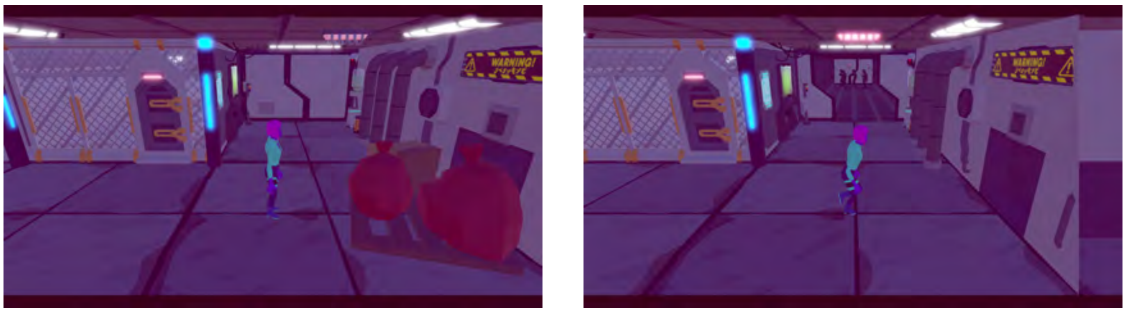

(b) End of the corridor.

Figure 6.20: Corridor area.

Once in the intersection, the player must use fire to burn a pile of objects that prevent the character from continuing her way. When this occurs, an alarm starts to sound while emitting a red light. At the same time, a door in the background opens revealing a set of guards. Therefore, it is hinted that the main character is escaping a prison against her will of whoever has imprisoned her. The main character will then make her way through a ventilation shaft, revealed after burning the pile of objects. This sequence is shown in Fig. 6.21.



(a) Before burning the pile of objects. **(b)** After burning the pile of objects.

Figure 6.21: Corridor intersection area.

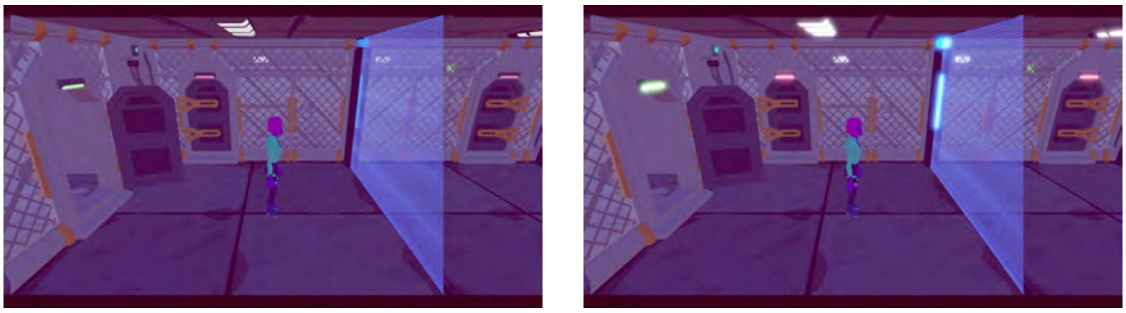
Finally, the level ends after the player reaches the end of the ventilation shaft. There, an empty room is located as a placeholder where a new area was going to be designed. However, this version of the game was discarded before this task was accomplished.

6.4.3 Post-Processing

A series of post-processing effects were used to get the desired cyberpunk look-and-feel.

6.4.3.1 Bloom

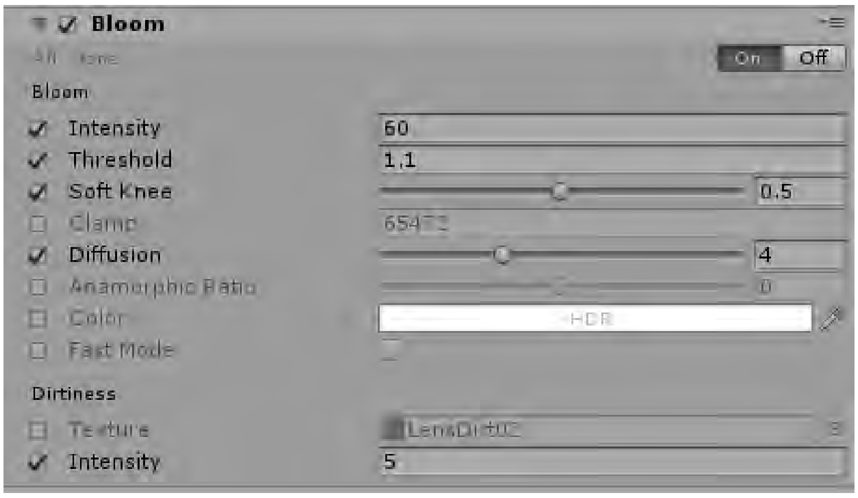
`Bloom` is a post-processing effect that aims to improve the look of bright areas and light sources by simulating glow. This is achieved by making the color of these bright zones bleed into neighboring pixels. Fig. 6.22 shows how this post-processing effect influences the look of the game.



(a) `Bloom` disabled. **(b)** `Bloom` enabled.

Figure 6.22: Influence of the `bloom` post-processing effect.

The values of the parameters for the effect were estimated by trial and error until the desired results were achieved. These values are shown in Fig. 6.23.

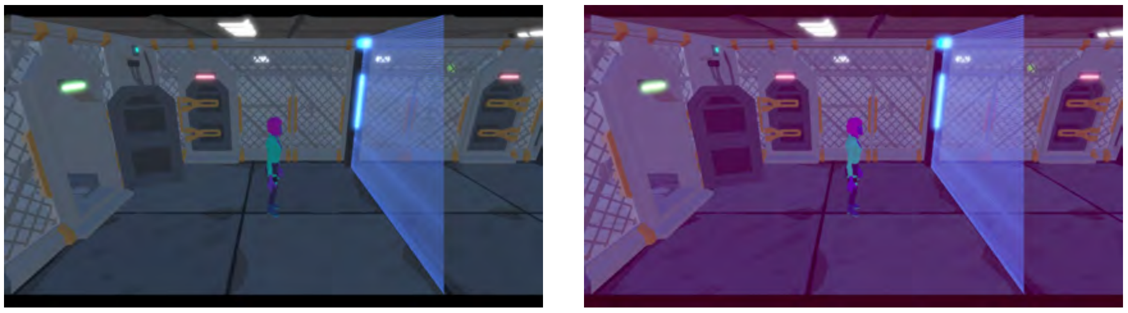


```
Bloom
All  None                              On  Off
Bloom
✓ Intensity            60
✓ Threshold            1.1
✓ Soft Knee            0.5
☐ Clamp                654[illegible]
✓ Diffusion            4
☐ Anamorphic Ratio     0
☐ Color                HDR
☐ Fast Mode
Dirtiness
☐ Texture              LensDirt0[illegible]
✓ Intensity            5
```

Figure 6.23: Values of the `bloom` post-processing effect.

6.4.3.2 Color Grading

In computer graphics, `color grading` is a post-processing effect that can be used to improve the color balance of the rendered image, as well as its luminance and contrast. It is often also used to implement a color correction filter that produces an adequate and pleasant appearance of the rendered scene. Fig. 6.24 shows how this post-processing effect influences the look of the game.



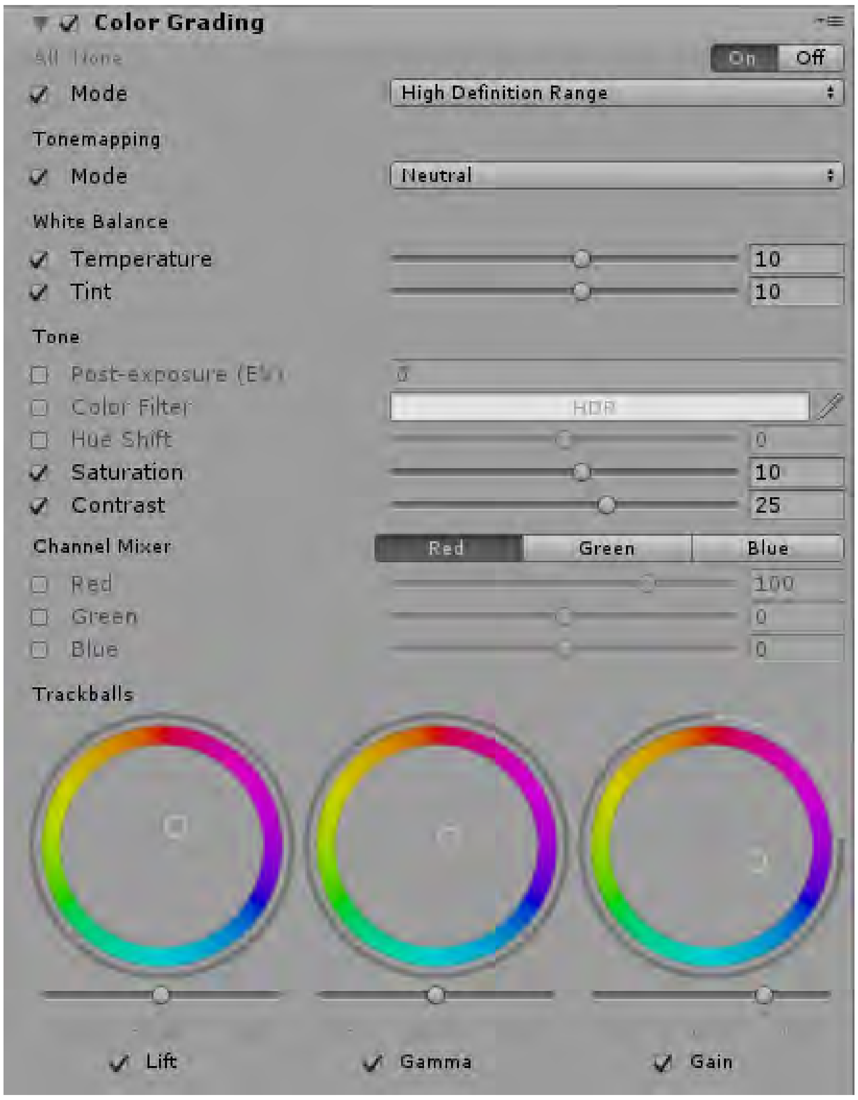
(a) `Color grading` disabled. (b) `Color grading` enabled.

Figure 6.24: Influence of the `color grading` post-processing effect.

To achieve an effect where the scene seemed under the influence of neon lighting, various parameters such as `tonemapping`, `hue`, `white balance`, `saturation`, `contrast`, `lift`, `gamma` and `gain` were adjusted to a set of specific values. However, the most relevant among them all were the last three: `lift`, `gamma` and `gain`. These values conferred the scene with the desired purple neon style supporting the cyberpunk thematic. In Fig. 6.25, all these values are showcased.

6.4.4 Menus and UI

As mentioned earlier, this early version of the game was a prototype discarded before finishing it. Therefore, a fully working menu was not implemented. However, since



✓ **Color Grading**

All None — On | Off

✓ Mode: High Definition Range

Tonemapping

✓ Mode: Neutral

White Balance

✓ Temperature: 10

✓ Tint: 10

Tone

☐ Post-exposure (EV): 0

☐ Color Filter: HDR

☐ Hue Shift: 0

✓ Saturation: 10

✓ Contrast: 25

Channel Mixer: Red | Green | Blue

☐ Red: 100

☐ Green: 0

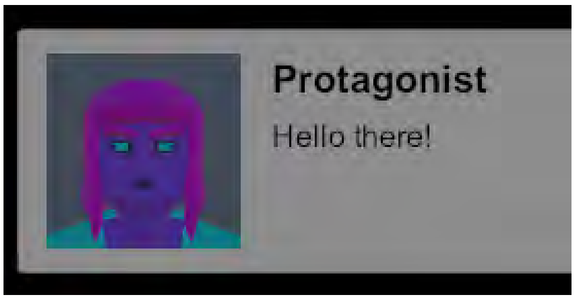
☐ Blue: 0

Trackballs

✓ Lift ✓ Gamma ✓ Gain

Figure 6.25: Values of the `color grading` post-processing effect.

it was planned to have dialogues between the main character and *The Hacker*, an early version of a dialogue system was implemented. This system is shown in Fig. 6.26.


Protagonist

Hello there!

Figure 6.26: Early version of the dialogue system.

6.4.5 Post-Mortem

After having implemented this prototype, it was time to take a critical perspective and analyze whether it had valuable potential or not. Once this analysis was performed, the master thesis authors realized that the game presented various flaws as it was.

First, the game interaction was far from perfect. The only action that the player could take was to click on designated areas and witness how the main character automatically walks once the right action has been performed. Second, as a consequence of the first problem, the game was not challenging. There were no losing conditions nor incorrect choices.

Having identified these problems, the team decided that a substantial change in the level design was required to fix them, following the agile principle "embrace the change" explained in Sec. 4.1.1. The level designed on this version was discarded and the team started to design a new level based on the discarded one, leading to a second version of the game.

6.5 Phase 5: Second Version of Under Surveillance

As explained in the previous section, in order to solve all the problems that the first version of *Under Surveillance* presented, a new version was developed. This remake does not present any substantial changes in terms of visual identity nor narrative. The efforts were instead focused on improving the level design.

6.5.1 Level Design

Two ideas were generated to solve the problems the level design from the first version of the game had.

First, increasing the complexity of the level design. The reason behind this idea was to naturally reduce the repetitiveness of the gameplay.

Second, adding more challenge to the gameplay. To accomplish this while maintaining the original core of the game in terms of interaction, it was decided to include more complex puzzles. While these puzzles can be solved just by clicking on the screen, they do require thinking to be completed.

Therefore, after implementing these two ideas into the game, the player needs to actively engage in a significant challenge to complete the game.

6.5.1.1 Area 1: The Prison

In this reworked area, the major change is a notable increase in the complexity of the environment. Not only it is populated with more game elements, but it also resembles a more realistic prison. Screenshots of the whole area are shown in Fig. 6.27.

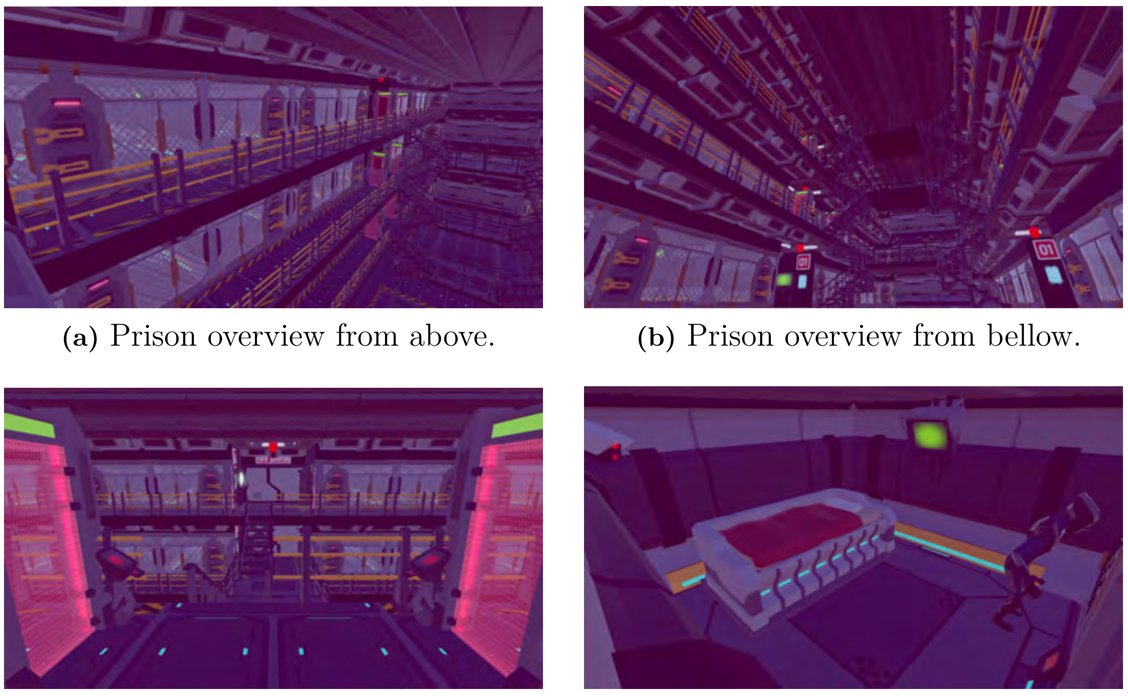
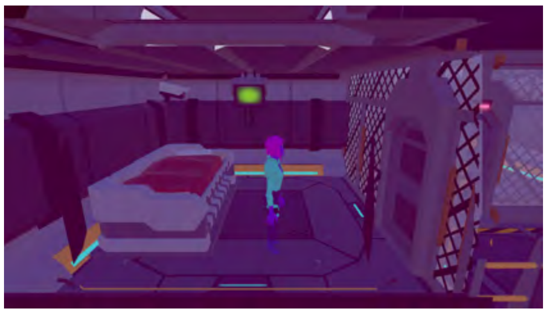

(a) Prison overview from above. **(b)** Prison overview from bellow. **(c)** Last floor of the prison. **(d)** Jail cell.

Figure 6.27: Area 1: The Prison.

From a structural perspective, the game starts similarly as it did in the previous version in this area. The main character is locked in a jail cell and *The Hacker* is supposed to help her escape. When this happens, the path leads to a corridor full of other jail cells and prisoners, as well as an electric barrier that the player must disable to proceed further in the level. This sequence is illustrated in Fig. 6.28.

Another important change in this area is that both the alarm effect and the set of guards appear when the player has disabled the electric barrier. When this happens,


(a) The main character imprisoned in the jail cell.

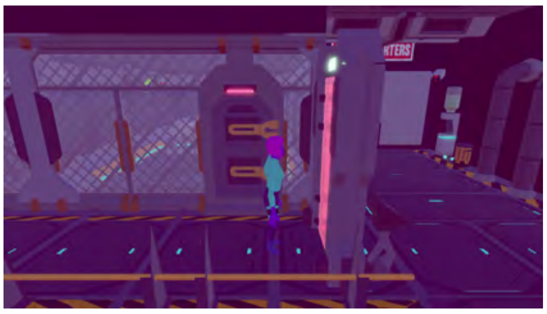

(b) The main character waiting for *The Hacker* to disable the electric barrier.

Figure 6.28: First part of the prison.

the main character takes an alternative route using the stairs located nearby. By doing this, the character moves one floor below in the prison. This sequence is shown in Fig. 6.29.

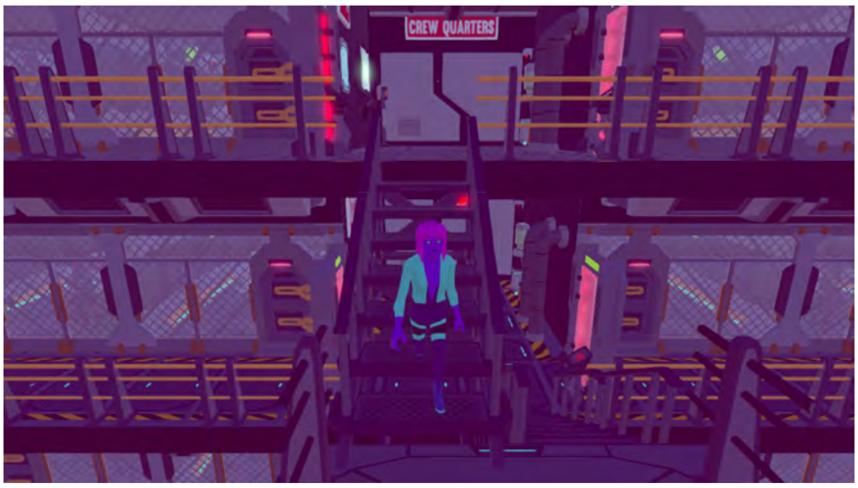


Figure 6.29: Main character moving downstairs.

Once in this new floor, the character moves towards a door located in the background, accessing to the second area of the level.

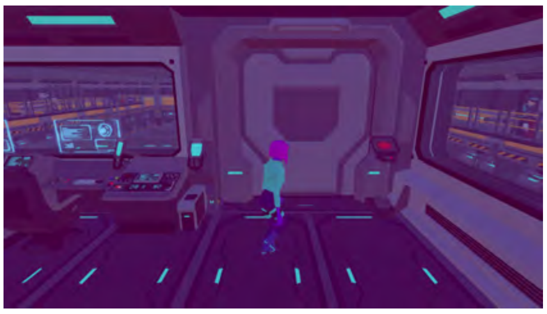
6.5.1.2 Area 2: The Storage Room

In this new area, the player is presented with a huge storage room. It is possible to observe two cranes, a set of huge metallic boxes and few workers on their computers. The cranes and the boxes form the puzzle that the player must solve to proceed further in the game. The workers, animated using *Mixamo* animations, are not a relevant part of the gameplay. Their presence fulfills the purpose of making the environment look more realistic. Screenshots of this area are presented in Fig. 6.30.

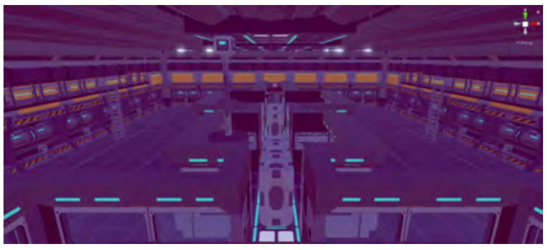
About the puzzle itself, it consists of two independent parts. In each of them, the player takes control of a crane that can push the metallic boxes. The goal of moving

(a) The control room.

(b) Security guard working.


(c) The main character leaving the control room.

(d) The main character approaching the puzzle.


(e) Overview of the puzzle.

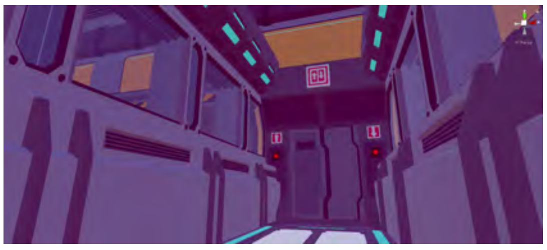
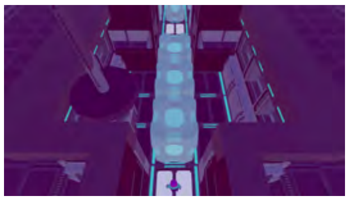
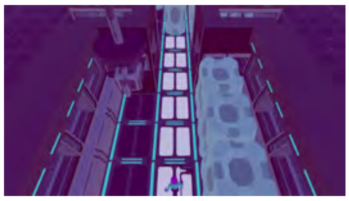

(f) Exit of the storage room.

Figure 6.30: Area 2: The Storage Room.

the boxes by using the crane is to create a safe path for the main character. The challenge lies in restricting the crane's movement. First, the crane nor the boxes can leave the small room in which they are located. Second, the crane cannot go through the boxes without pushing them. Finally, boxes can push other boxes. Screenshots of this puzzle are shown in Fig. 6.31.


(a) First part of the puzzle.


(b) Solution to the first part of the puzzle.

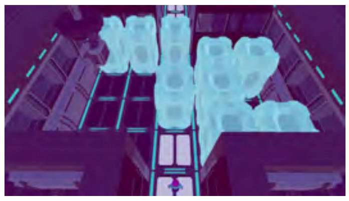

(c) Second and last part of the puzzle.

Figure 6.31: Crane and metallic boxes puzzle.

The process of designing the puzzle in *Unity* is showcased in the following video: `https://youtu.be/PbOtRsOOtYc` [65].

The main challenge behind implementing this puzzle consists on coding the crane movement behavior. The algorithms related to collisions and forces are automatically handled by *Unity*'s physics engine. The `PuzzleCraneMovement` script, constitutes the most important part of the crane movement behavior.

Once the player solves this puzzle, the main character can walk towards an elevator that would lead to the next area. However, this version of the game was discarded before further proceeding in the development.

6.5.2 Post-Mortem

After having developed a working game prototype in which the planned changes were applied, it was again time to perform another critical analysis, checking whether or not the problems were solved. The master thesis authors then realized that the absence of challenge was indeed solved by including the puzzle. However, new problems were detected.

First, in terms of interaction, the team did not detect any major improvements. While the authors thought that with a more complex environment the interaction possibilities would increase, this was not the case. Other than in the puzzle, the player was still supposed to only click on designated areas, while observing how the main character automatically moves. Therefore, while the new environment was probably superior in terms of its visuals, the interaction with the environment did not improve accordingly.

Second, another major issue was that the game did not truly provide a physics-based experience. Even if electricity was used through the whole level, its usage just consisted on interacting with specific devices. If the electricity was removed and replaced with simple mouse clicks, there would be no difference in the gameplay.

Finally, the master thesis authors concluded that the implemented visual identity was not up to their quality standards. A cyberpunk atmosphere was achieved, but it did not quite look as pleasant as the authors envisioned. A particularly problematic aspect of these visuals was the lack of contrast. The whole scene was affected by the purple color achieved due to the color grading post-processing effect, resulting in an image that looked plain. Furthermore, given the big amount of game elements, no elements could stand out from the environment, not even those that were interactive, and even worse, neither the main character.

Once again, applying the "embrace the change" principle explained in Sec. 4.1.1, it was decided to discard this version of the game and start again.

6.6 Phase 6: Final Version of Under Surveillance

With all the lessons learned from the previous two versions, a third and final version of the game was developed. The core idea in this version was to focus on the interaction and the gameplay, using a strong visual identity to support these two while all the design and implementation overall was simplified.

The visuals were also simplified by reducing the environment elements to just those that provided value to the gameplay. The color palette complexity was also reduced, by having a gray-scale colored environment, with very sporadic use of color to support interaction and provide better feedback. This is achieved by utilizing color to provide contrast, particularly to the main character and the most relevant elements of the environment.

To increase the interaction possibilities, it was decided that every environment element should be responsive to the player's actions. For this to work, the main character's automatic movement was discarded. The reason behind this was that the master thesis authors considered that an interactive environment should be explored as freely as possible. The player must able to decide when and where to interact with the environment so that these interactions feel real and not scripted.

6.6.1 Narrative Design

In addition to the simplification process followed throughout the final version of Under Surveillance, the narrative was revisited for the first time since Sec. 6.1.2 and made easier to follow while keeping in mind the works that inspired *Under Surveillance*, presented in Sec. 2.2, in the first place. This simplification meant to remove from the development a lot of narrative elements such as the importance of a powerful A.I. ruling the world and the personification of the player as *The Hacker*.

Therefore, *Under Surveillance* is a 2.5D physics-based puzzle adventure video game in which the player follows a mysterious robot who can use and manipulate electricity and water to solve a series of puzzles while progressing further in an unknown dystopian world surrounded by a noir and eerie post-apocalyptic atmosphere. In this world, a mass surveillance state is present. Players unveil this world as they progress, collecting little pieces of information that are spread out everywhere in the world.

Thanks to this new setting, the player is not overwhelmed by a complex story that could have taken them away from the core of the game, its unique way of playing.

6.6.2 Software Architecture

In relation to the implementation, Fig. 6.32 presents some of the script classes and the relationships between them. It is important to note that only the most relevant classes are illustrated in this diagram, while many are omitted due to their relatively small overall importance.

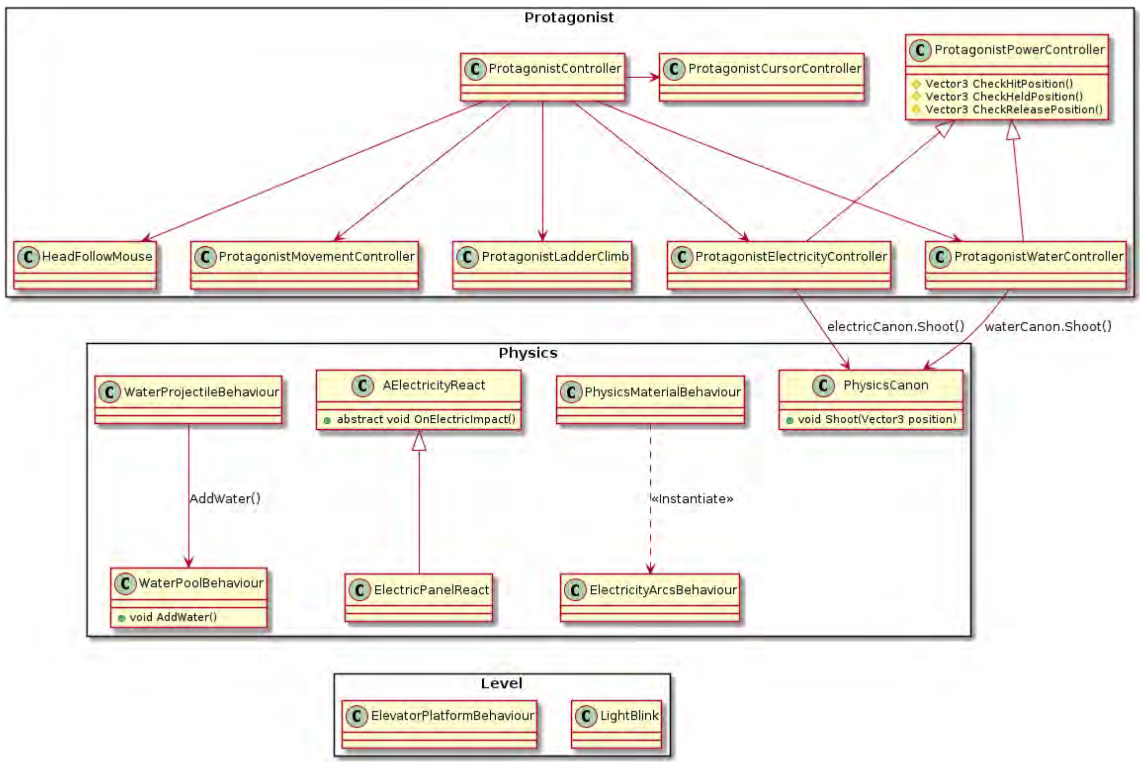

Protagonist
ProtagonistController
ProtagonistCursorController
ProtagonistPowerController
Vector3 CheckHitPosition()
Vector3 CheckHeldPosition()
Vector3 CheckReleasePosition()
HeadFollowMouse
ProtagonistMovementController
ProtagonistLadderClimb
ProtagonistElectricityController
ProtagonistWaterController
electricCanon.Shoot()
waterCanon.Shoot()
Physics
WaterProjectileBehaviour
AElectricityReact
abstract void OnElectricImpact()
PhysicsMaterialBehaviour
PhysicsCanon
void Shoot(Vector3 position)
AddWater()
«Instantiate»
WaterPoolBehaviour
void AddWater()
ElectricPanelReact
ElectricityArcsBehaviour
Level
ElevatorPlatformBehaviour
LightBlink

Figure 6.32: Class diagram of *Under Surveillance*.

The classes placed inside the `Protagonist` are those attached to the main character. The ones placed inside `Physics` are related to the electricity and water behaviors. Finally, those placed inside `Level` are behaviors attached to objects placed throughout the level. All these scripts will be further explained in the following sections.

6.6.3 Character Design

First, the concept of *The Hacker* was removed. Moreover, unlike the other versions in which the main character was a cyberpunk-stylized woman robot, in this version it was replaced by a robot that every player can feel identified with, regardless of their gender[3]. In Fig. 6.33, the new character is shown.

[3]More information behind this decision is discussed in Sec. 8.3.

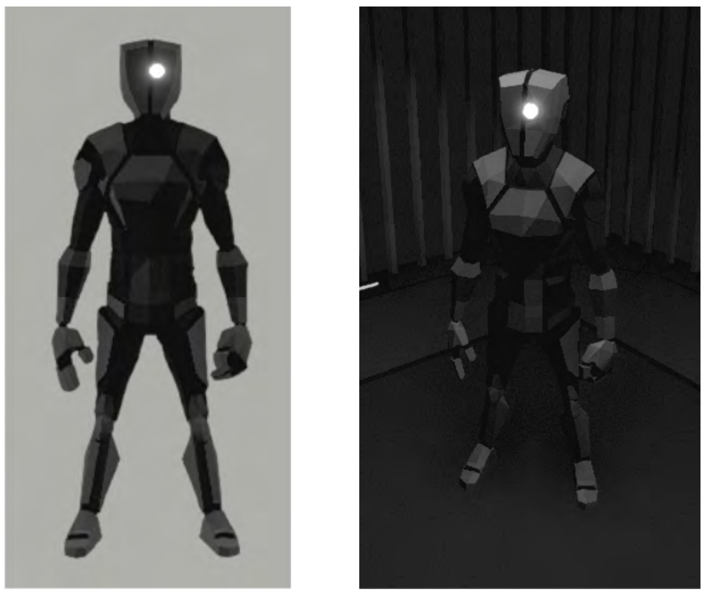


Figure 6.33: The new main character of the last version of *Under Surveillance*.

The new main character is part of the *Polygon - Sci-Fi City Pack* asset package. Since the mesh of the character was already available to the master thesis authors, all the efforts went into animating and scripting the character.

6.6.3.1 Animations

All the animations were acquired from *Mixamo*. To achieve smooth and pleasant animations on the main character in every possible situation present in the game, many different animation clips have been included. All of these clips and the transitions between them are controlled by the character's animator controller, shown in Fig. 6.34.

Each of the nodes from Fig. 6.34 represents a character state and its corresponding animation clip. Likewise, each of the edges connecting these nodes represents a possible transition between these states. Each of these transitions can be configured to either automatically happen after a given time, or happen when a particular condition is met.

The `Entry` node represents the start point of the animator state machine. From this point, the system automatically transitions to the adjacent state: `SittingIdle`. Therefore, this is the first state displayed by the character when the game starts. The transition to the next state, `StandingUp`, is configured so that it is only performed when the player clicks on the character. The subsequent transitions are performed automatically until the state `Idle` is reached.

From the gameplay perspective, this means that there is an initial cutscene. The player will initially see a sat character. Then, when they click on this character it will stand up and look around. Finally, when the character reaches the `Idle` state, the cutscene is finished. The player is then allowed to move and use the character's

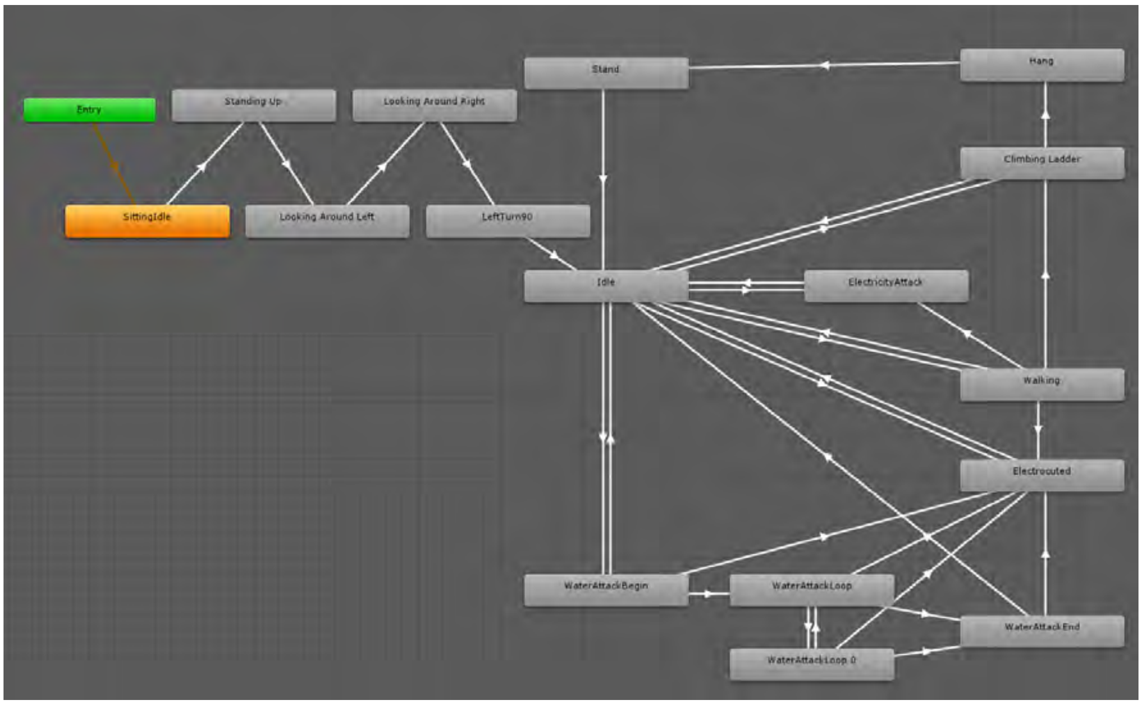


Figure 6.34: Main character's animator controller.

abilities. Screenshots of this cutscene are shown in Fig. 6.35.

From the `Idle` state, it is possible to reach many other states.

First, the `Walking` state is used when the character moves, displaying a visually appropriate animation in which the character takes steps.

Second, the `ElectricityAttack` state is used when the character throws an electricity ball, placing its hands near its chest and making a throw movement. Likewise, the `WaterAttackBegin`, `WaterAttackLoop`, `WaterAttackLoop 0` and `WaterAttackEnd` states are used when the character throws water. More than one state is required because water can be thrown for an infinite amount of time, requiring an animation that can be looped.

Third, the `ClimbingLadder` state is used when the character is climbing a ladder. The subsequent states `Hang` and `Stand` are used when the top of the ladder is reached.

Finally, the `Electrocuted` state becomes relevant when the character falls inside a pool full of water. In this situation, the robot will suffer an electrical shock, which is shown through the corresponding animation of the state.

6.6.3.2 Scripting

For the main character to behave appropriately in every situation of the game, several scripts are attached to its game object. The main idea when these scripts were written, was to have the code as organized and decoupled as possible. Therefore, instead of writing a single script that implements all the main character's functionality, multiple smaller scripts with specialized functionalities have been written. The

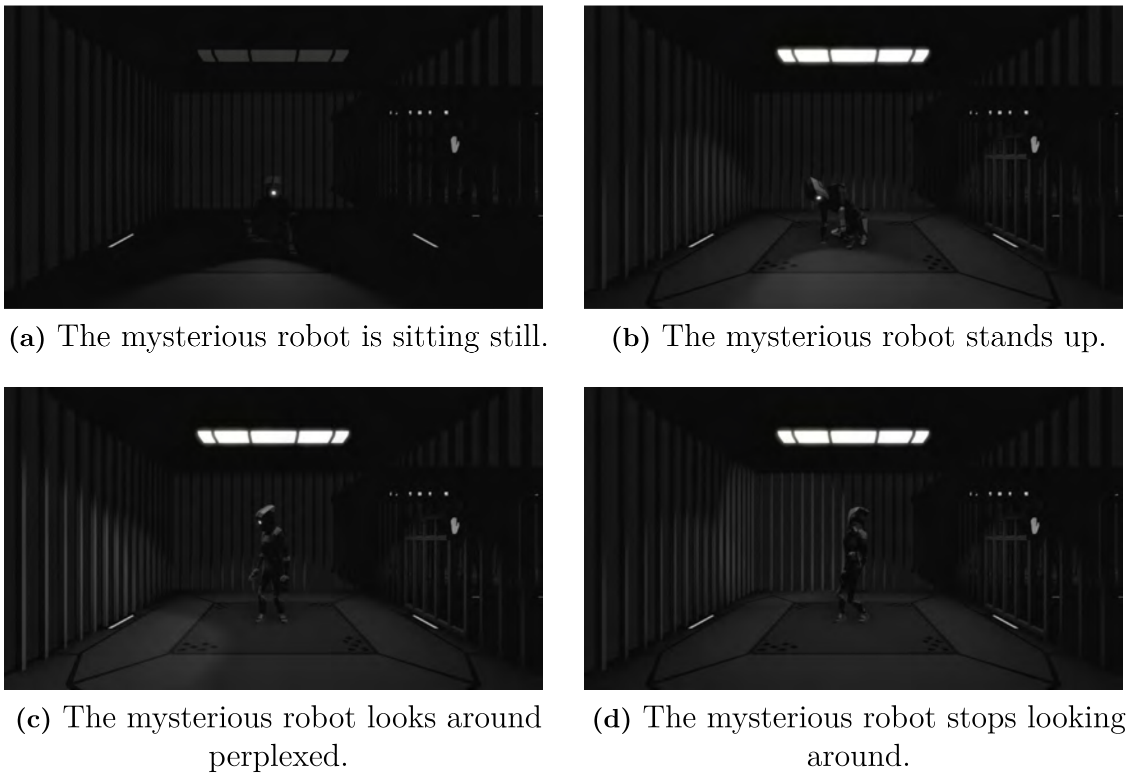

(a) The mysterious robot is sitting still.
(b) The mysterious robot stands up.
(c) The mysterious robot looks around perplexed.
(d) The mysterious robot stops looking around.

Figure 6.35: Initial cutscene at the beginning of *Under Surveillance*.

most important scripts related to the main character are:

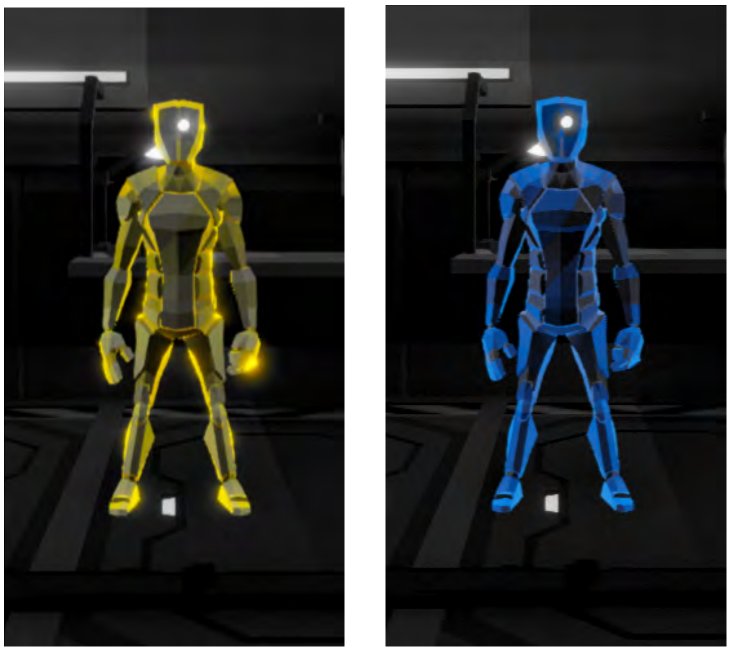
- `ProtagonistController`: This script is responsible for keeping state variables shared by the other scripts as well as enabling or disabling them when it is appropriate. Therefore, it acts more like a facade or a dispatcher that helps to keep the code organized than implementing actual functionality.
- `HeadFollowMouse`: This script is responsible for making the main character look at the mouse position by rotating its head. This feature was implemented to help the player figure out that the mouse can be used to interact with the game.
- `ProtagonistMovementController`: This script is responsible for making the main character move left and right. It sets the horizontal velocity of the game object according to the player's input and rotates the character to face the appropriate direction.
- `ProtagonistLadderClimb`: This script is responsible for making the main character climb ladders. It sets the vertical velocity of the game object according to the player's input and controls the animations displayed when climbing ladders.

There are also two additional scripts attached to the protagonist's game object: `ProtagonistElectricityController` and `ProtagonistWaterController`. These will be explained in detail in Sec. 6.6.4, since they are related to the game's physics-based elements.

6.6.4 Physics

As mentioned earlier, the character has two physics-based special abilities: throwing electricity and water. These abilities are not presented at the same time to the player, because that might overwhelm them in terms of the number of interactions. Instead, they are introduced in different parts of the level, so the player becomes familiar with the first ability before introducing the second one. This is explained in more detail in Sec. 6.6.5.

Since both abilities are available to the player at the same time, it is necessary to let them choose which one to use. This problem has been solved by having two different states on the character: electricity state and water state. The player can swap between these two states by clicking on the main character. It was also crucial to implement a way to provide feedback so that the player is aware of the current character's state. In this regard, it was decided not to have any HUD. Instead, this information is provided by the character itself, as illustrated in Fig. 6.36, by showing a different appearance depending on its state.



(a) The robot's electricity state.

(b) The robot's water state.

Figure 6.36: Appearances of the main character depending on its state.

In relation to the code written to implement the physics' elements and their behaviors, both electricity and water share some basic behavior. For instance, both abilities are activated when the player clicks anywhere in the game world. Also, the projectile is always shot towards the position the player clicked, both if it is electricity or water.

Consequently, this common behavior has been written in separate scripts, to allow reuse both from the electricity and water behaviors. These scripts are:

- `PhysicsCanon`: This is script is responsible for instantiating the physics' element projectile (electricity or water) and setting its direction and velocity depending on the target position.
- `ProtagonistPowerController`: This script is a generalization of the scripts `ProtagonistElectricityController` and `ProtagonistWaterController`, explained in Sec. 6.6.4.1 and Sec. 6.6.4.2, respectively. It provides both these scripts with convenient methods to check whether the mouse has been clicked, released or it is being held, as well as its position. Also, it encapsulates their common fields and behaviors.

6.6.4.1 Electricity

The first special ability that the player is given is throwing electricity. By clicking anywhere in the game world, the main character will shoot an electrical ball that will move towards the clicked position. The *Polygon Arsenal* asset package was used to achieve this effect. Once the ball hits an object, a reaction depending on this object's properties is triggered. In the following paragraphs it is going to be first analyzed the electricity projectile and its behavior, and then the reactions of the environment when the projectile collides with any object.

The script responsible for managing the creation of each electricity projectile is `ProtagonistElectricityController`. When it detects that the click has been performed, it starts the adequate main character's animation, while also invokes the electricity's `PhysicsCanon` at the appropriate moment. This results in the main character visually throwing an electrical ball, as shown in Fig. 6.37.

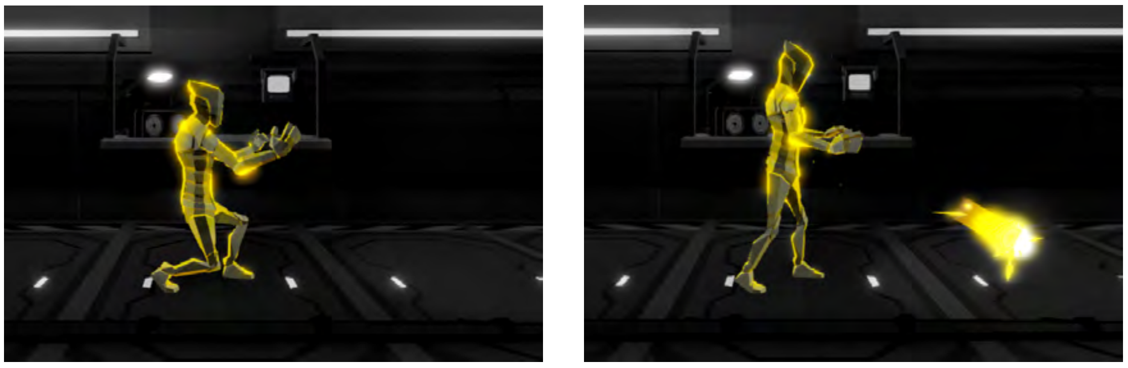
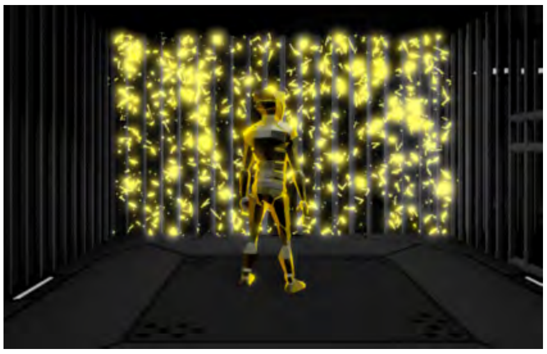

(a) Beginning of the animation. **(b)** End of the animation.

Figure 6.37: Main character's electricity throw animation.

When the electric ball hits any object, there are two possibilities: either the object can conduct electricity or it cannot. The triggered reaction in these two cases is completely different. If the object can conduct electricity, then electrical arcs and sparks will arise from every part of the object, eventually fading out. On the other hand, if the object cannot conduct electricity a small electrical explosion is the only visible effect. These two reactions are shown in Fig. 6.38.


(a) Reaction on conductive objects.

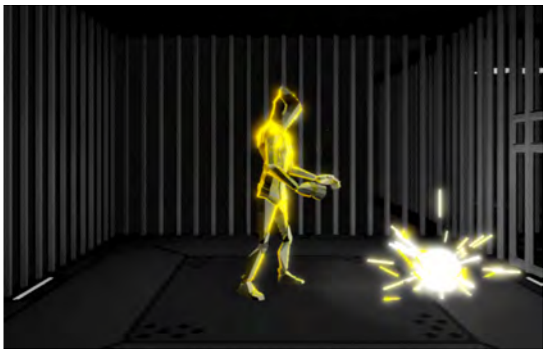

(b) Reaction on non-conductive objects.

Figure 6.38: Possible reactions when an electric ball hits any object.

The script `PhysicsMaterialBehaviour` is responsible for instantiating the electrical arcs and sparks if the object is made of a conductive material. This is achieved by instantiating an object with the `EletricityArcsBehaviour` script attached to it. The `EletricityArcsBehaviour` script generates the particle systems needed to display the arcs and sparks. It does so by obtaining the hit object's collider and using it to determine the shape and size of the particle systems' emitters.

Electricity can also be used to interact with some devices placed through the level, such as elevators' terminals. Code extensibility and re-usability was kept in mind when implementing this feature. Therefore, an abstract class, namely `AElectricityReact` was created. Any script extending this class needs to implement the `OnElectricImpact()` method. If a game object has attached any script extending this class, when an electric projectile hits it, its `OnElectricImpact()` method will be automatically called.

Finally, electricity can be also used to turn on electrical panels, controlled by the `ElectricPanelReact` script. These panels have wires attached, connecting them to a number of devices. When an electrical panel is turned on, electricity will flow through its wires, consequently enabling all the connected devices. This behavior is shown in Fig. 6.39.

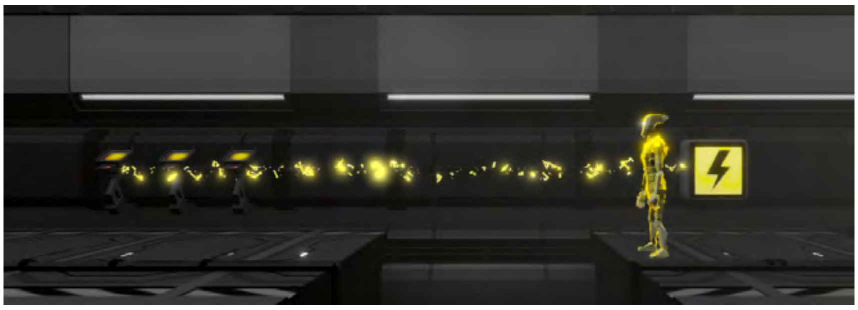


Figure 6.39: Electrical panel turned on.

Panels can also push electricity through cut wires if both ends are connected through the water contained in a pool. This behavior is shown in Fig. 6.40.

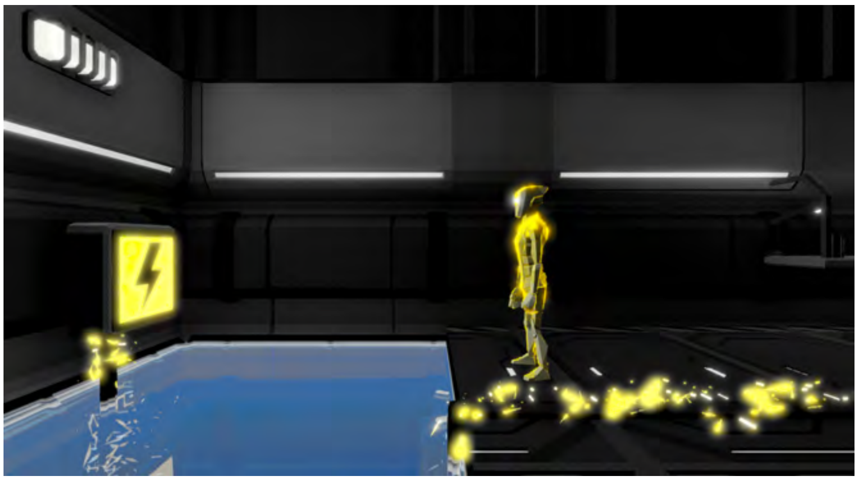
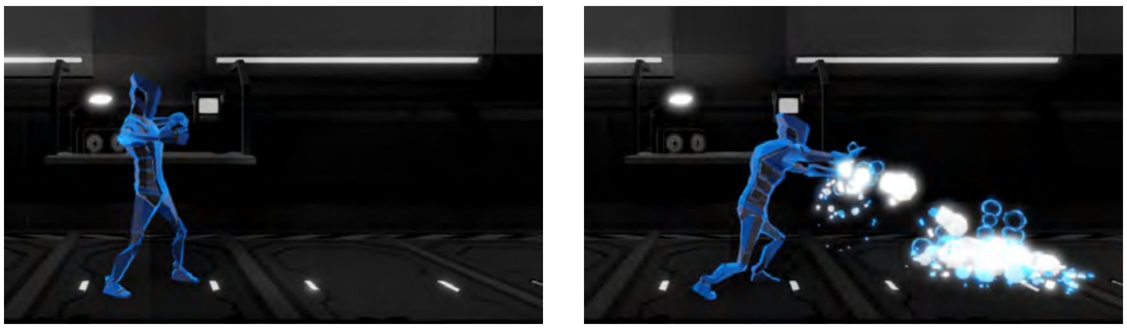


Figure 6.40: Water conducting electricity.

6.6.4.2 Water

The second special ability that the player is given is throwing water. By holding the mouse button anywhere in the game world, the main character will shoot a stream of water made out of a number droplets, obtained from the *Polygon Arsenal* asset package. These droplets will move towards the clicked position while being affected by gravity, consequently describing a parabolic trajectory. Once each of these droplets hits any object a splash effect is produced. Furthermore, if the splash is produced inside a cavity the water will accumulate, forming a pool. In the following paragraphs, it is going to be first analyzed the water stream and its behavior, and then how water can interact with the environment.

The script responsible for managing the creation of each of the water droplets is `ProtagonistWaterController`. When it detects that the mouse button is being held, it starts the adequate main character's animation, while also invokes the water's `PhysicsCanon` at the appropriate moment. This results in the main character visually throwing a water stream, as shown in Fig. 6.41.



(a) Beginning of the animation. **(b)** Loop of the animation.

Figure 6.41: Main character's water throw animation.

When each water droplet hits any object it is checked whether or not that happened in a concavity. Concave areas hereinafter referred to as pool areas, are manually defined by using trigger volumes. The script `WaterProjectileBehaviour` checks whether each water droplet collided inside a pool area. If that condition is met, then water is added to the pool area. Furthermore, a water ripple effect is generated on the surface of the pool area.

The script `WaterPoolBehaviour` is responsible for managing the pool areas. It provides the public method `AddWater()`, that raises the water level by an amount that depends on the pool's volume.

All these scripts allow the player to put water inside these pool areas, as shown in Fig. 6.42. However, some technical aspects of the water have not been implemented by the authors. Instead, the *Water 2D Tool* asset package has been used. This asset already implements some sophisticated water physics, such as ripple generation and buoyancy.

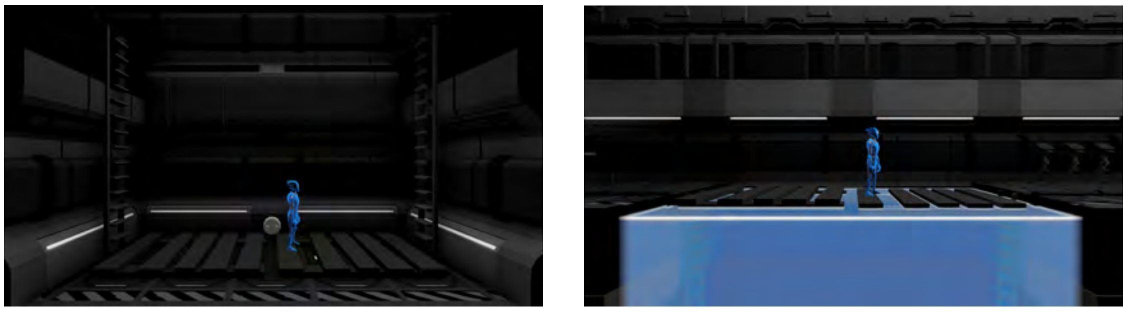

(a) Empty pool area. (b) Pool area with water.

Figure 6.42: Pool area.

6.6.4.3 Fire

Unfortunately, the design and implementation of the fire were discarded at this phase of the development given the short remaining amount of time by this point.

6.6.5 Level Design

The main level of *Under Surveillance* is divided into four different areas: escaping the jail cell, corridor, elevator and swimming pools.

6.6.5.1 Area 1: Escaping the Jail Cell

The game starts with the main character sat down inside a jail. As mentioned in Sec. 6.6.3.1, when the player clicks on the main character a cutscene is played and then the player can move and interact with the environment.

Since the main character is inside a jail cell, the interaction possibilities are quite limited. This helps the player to get familiar with the game mechanics (move and throw electricity) before having to perform any complex action or be overwhelmed with a lot of environment elements. The only way to escape the jail is by throwing electricity to its door's lock and then walking out, which forces the player to use both these mechanics in a simple scenario.

Furthermore, the jail itself contains objects that act in the three main possible ways when hit by electricity. These possible reactions are: conducting electricity (performed by the bars of the jail), not conducting electricity (performed by the floor and ceiling of the jail) and having a special reaction to electricity (the lamp that blinks when electricity it receives electricity, and the door lock that opens). The player is not forced to perform most of these actions. However, their presence feels gratifying for explorer players [66] that will try to interact with every element and expect a consistent reaction.

Fig. 6.43 shows a few screenshots from this area.

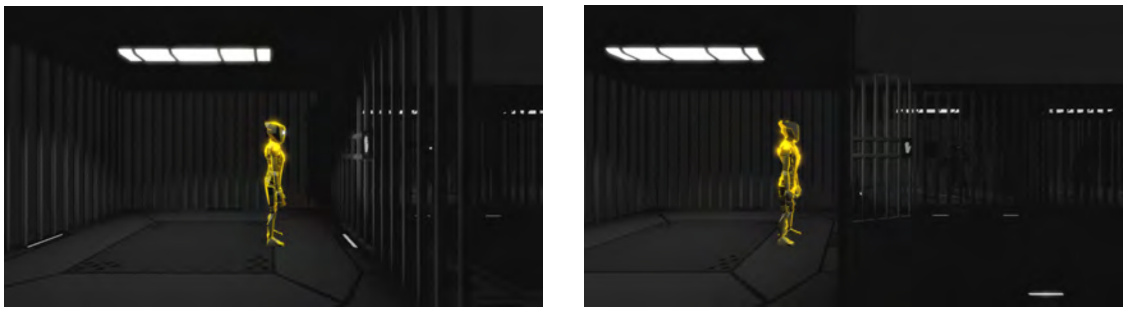

(a) The jail door is locked. **(b)** The player has unlocked the jail door.

Figure 6.43: Screenshots of the first area of the level.

In relation to the scripts related to this area, there is one behavior worth to be mentioned.

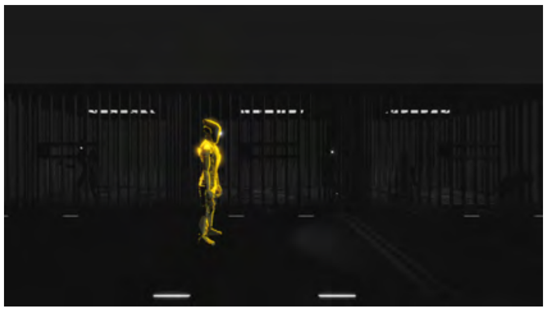
Before the initial character's cutscene is played, the light placed on the ceiling of the jail blinks while emitting an electric sound effect. To have both the sound and the light synchronized, it is the sound that drives the light's intensity value. During the light blink animations, the loudness of the sound effect is constantly checked, and the light intensity is adjusted according to it. This behavior is implemented in the `LightBlink` script.

6.6.5.2 Area 2: Corridor

After the player leaves the jail, a corridor is presented. It is assumed that the player has learned how to interact with the character and the environment in the previous area. Thus, this new area is more focused on quietly introducing the narrative to the player. In this corridor it is possible to witness other prisoners in their jails, therefore providing hints on the dystopian nature of the game world.

Another key element in this area is the presence of a radio outside these jails. This radio is turned on, suggesting that beings other than the prisoners were populating that area not a long time ago.

Fig. 6.44 shows a few screenshots from this area.


(a) Prisoners in the background.

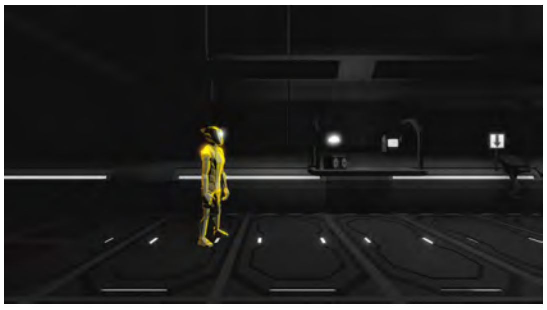

(b) Radio in the background.

Figure 6.44: Screenshots of the second area of the level.

Despite this, the main character can still use its throw electricity ability to interact with the environment. Conductive and non-conductive elements are present in this area. Furthermore, the radio generates noise audio when hit by electricity.

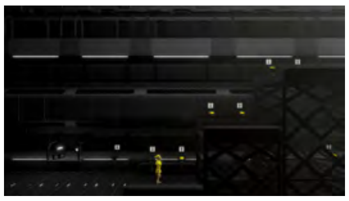
6.6.5.3 Area 3: Elevators

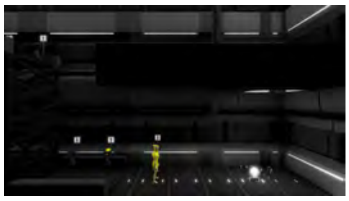
After the player is familiarized with the game mechanics as well as the game world, the first puzzle is introduced. In this area, the player must use three elevators to reach a platform placed in a high position.

The elevators can be interacted by the player by throwing electricity at their terminals. These terminals have a yellow emissive color that matches the electricity color. When the player throws electricity at a terminal, they will make the elevator go either up or down one step. The player must take advantage of this behavior to reach the aforementioned platform. This behavior is implemented in the `ElevatorPlatformBehaviour` script. Furthermore, to gratify explorer players there is a collectible secret that can be accessed by discovering an alternative route using these elevators.

Finally, when the player reaches the platform located on the top, they must climb a ladder that leads to a ventilation shaft, marking the end of this area.

Fig. 6.45 shows a few screenshots from the area.


(a) Bottom area.


(b) Secret area.

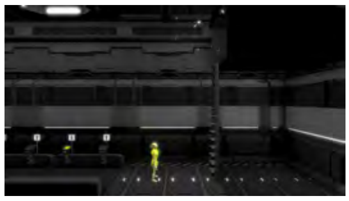

(c) Top area.

Figure 6.45: Screenshots of the third area of the level.

6.6.5.4 Area 4: Swimming Pools

After the player goes through the ventilation shaft, they reach the fourth and final area. In this area, the main character is provided with the ability to throw water. The puzzle presented in this area has to be solved by using both electricity and water. To solve it, the player must fill all the pools with water, so they conduct electricity from the panel located on the left part of the area, to the elevator panels located in the right part. By doing this, the elevator can be used to leave the area and complete the game.

The puzzle is designed so it can be completed by players with any skill level.

If the player walks to the right, the main character will eventually fall inside a pool that contains a pallet with no obvious way to get out. Thus, the player will try to interact with the environment in every possible way, and will eventually throw water. When this happens, the water level inside the pool goes up, and so does the pallet, consequently pushing the player upwards. With this design, it is guaranteed that after the player leaves that pool, they will understand that water can fill concave areas.

If the player walks to the left, they will see a pool filled with water as well as a panel with a lightning symbol. This symbol will hint the player to throw electricity at that item. By doing so, the panel starts to generate electricity that is conducted through the pool's water. This phenomenon is visually perceptible by the electric sparks that indicate that electricity is flowing. Consequently, the player will learn that water can conduct electricity.

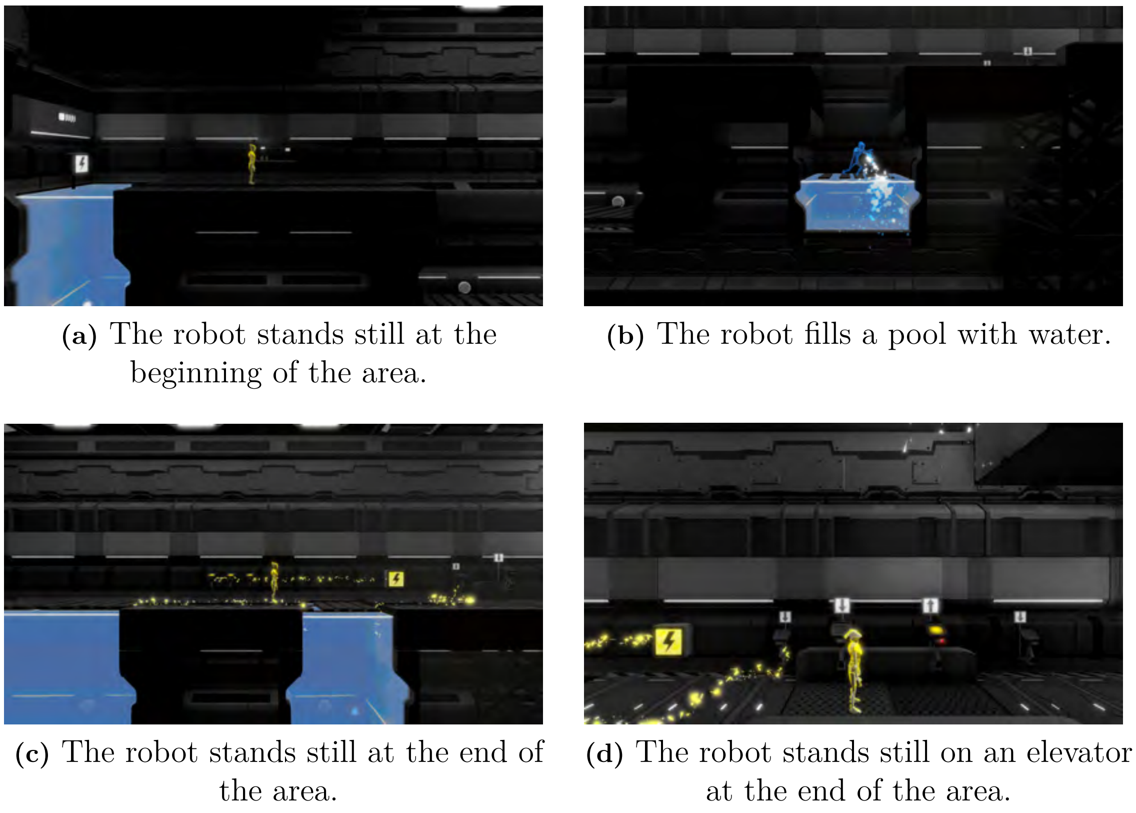
With these two pieces of knowledge, and given the fact that it is possible to see an electric wire that goes through all the pools, the player can infer that the objective of the puzzle is to conduct electricity through the pools, thus reaching the solution after some deductive thinking. However, to gratify more explorer players there is a secret hidden in the left-most pool. To collect it, the player must also deduce that pools can be emptied, using the terminals located at the center of the level, that can be used to open the drains located in each pool.

Fig. 6.46 shows a few screenshots from the area.

6.6.6 Lighting

The main character has a light in the head that enlightens the area to which it is looking at all times. Since this character is always looking in the direction of the mouse cursor, this allows the player to illuminate any part of the environment organically by moving the cursor over it. Fig. 6.47 shows this feature.

Furthermore, the *Aura - Volumetric Lighting* asset package was used to simulate scattered lighting. This light source is placed in the ventilation shaft that the player has to reach in order to complete the level. This is shown in Fig. 6.48. It is important to note that the effect has been exaggerated so that it is visible in the screenshot.


(a) The robot stands still at the beginning of the area.

(b) The robot fills a pool with water.

(c) The robot stands still at the end of the area.

(d) The robot stands still on an elevator at the end of the area.

Figure 6.46: Screenshots of the fourth area of the level.

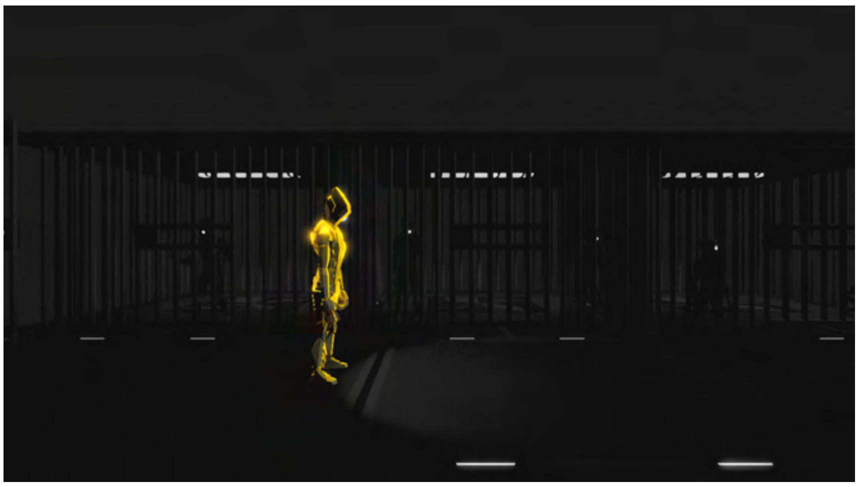


Figure 6.47: The main character enlightening the floor.

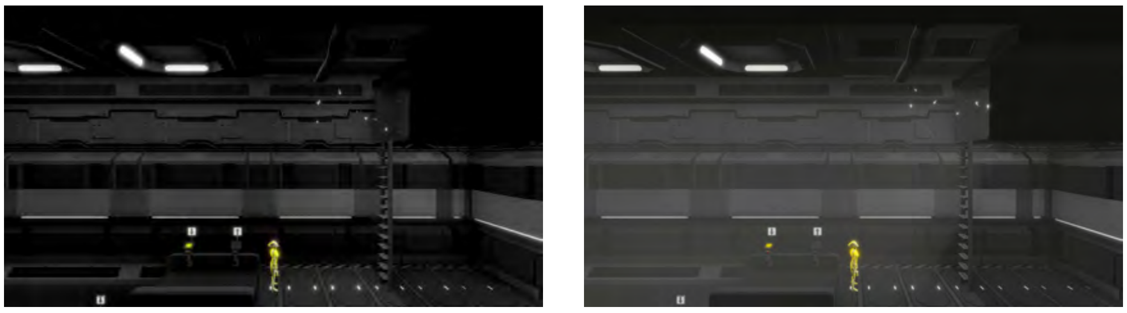

(a) Scattered lighting disabled. (b) Scattered lighting enabled.

Figure 6.48: Scattered lighting comparison.

6.6.7 Post-Processing

A series of post-processing effects were used to get the desired look-and-feel.

6.6.7.1 Ambient Occlusion

In the real world, some areas such as corners and other regions located between close surfaces tend to block light and look darker. In the context of computer graphics, `ambient occlusion` is an effect that simulates this phenomenon by occluding ambient light by some factor in specific areas of the screen. Ambient occlusion has been used as a post-processing effect in *Under Surveillance* to create a look of realism in lighting and emphasize the shapes of the different meshes. Fig. 6.49 shows how this post-processing effect influences the look of the game.

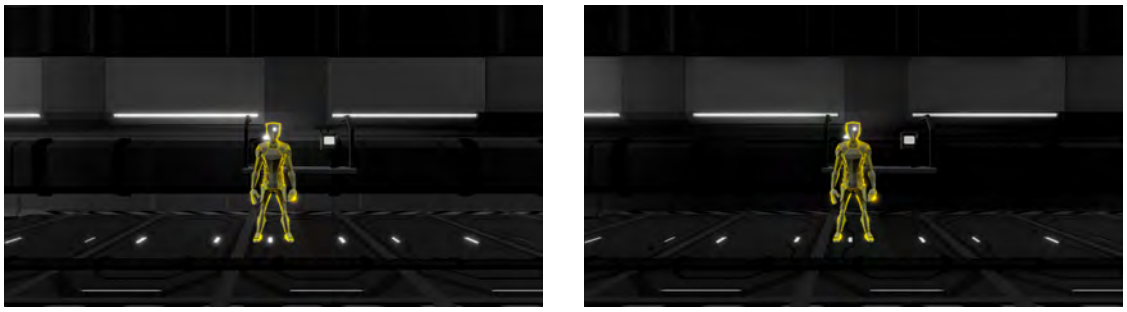
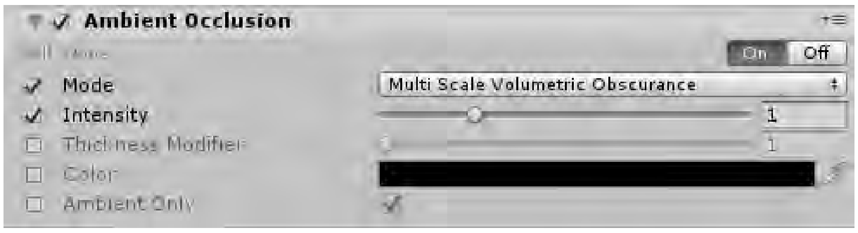

(a) `Ambient occlusion` disabled. (b) `Ambient occlusion` enabled.

Figure 6.49: Influence of the `ambient occlusion` post-processing effect.

In order to achieve the desired result, several parameters need to be set to appropriate values. For this effect in particular, these values were found by following an iterative process of trial and error, stopping when the output resulted appealing to both authors. Fig. 6.50 shows the final values of these parameters.

6.6.7.2 Bloom

Fig. 6.51 shows how `bloom` influences the look of the game in this version.

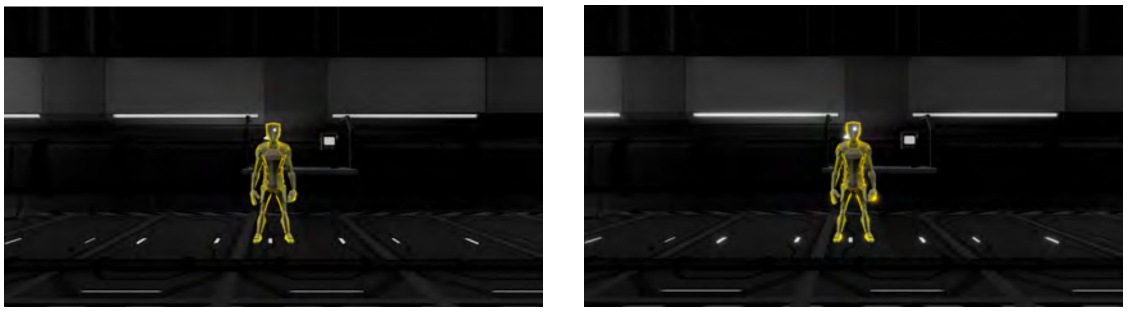


Ambient Occlusion

On Off

✓	Mode	Multi Scale Volumetric Obscurance
✓	Intensity	1
☐	Thickness Modifier	1
☐	Color	■
☐	Ambient Only	✓

Figure 6.50: Values of the `ambient occlusion` post-processing effect.

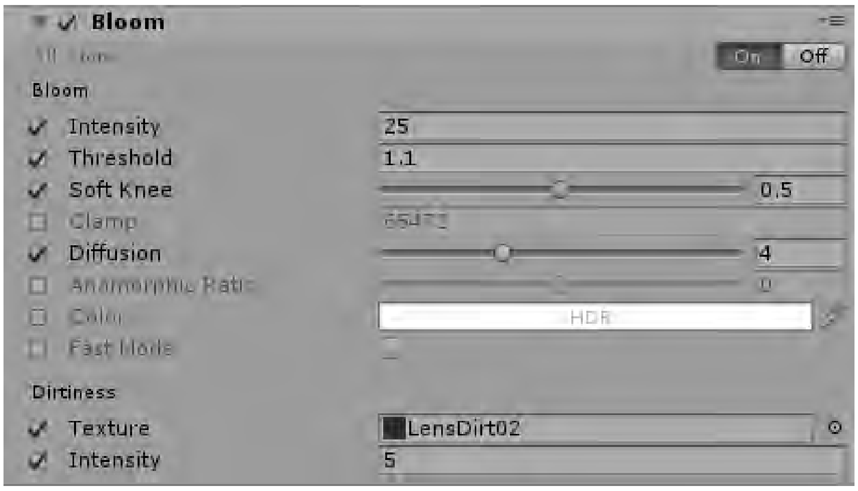


(a) `Bloom` disabled.

(b) `Bloom` enabled.

Figure 6.51: Influence of the `bloom` post-processing effect.

The parameters values of the effect were estimated by trial and error until a satisfactory result was achieved. These values are shown in Fig. 6.52.



Bloom

On Off

Bloom

✓	Intensity	25
✓	Threshold	1.1
✓	Soft Knee	0.5
☐	Clamp	65472
✓	Diffusion	4
☐	Anamorphic Ratio	0
☐	Color	HDR
☐	Fast Mode	

Dirtiness

✓	Texture	LensDirt02
✓	Intensity	5

Figure 6.52: Values of the `bloom` post-processing effect.

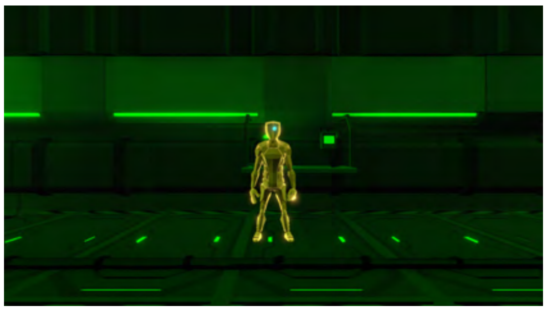
6.6.7.3 Color Grading

Although several options were modified to achieve the adequate `color grading` post-processing effect for the game, such as its `tonemapping`, `white balance`, `tone`, `channel mixer` and `grading curves`, only the latter are going to be discussed. The reason behind this is because the modification of the `grading curves` has the biggest impact on *Under Surveillance*'s `color grading`.

The `grading curves` within the `color grading` allowed to filter the hue of the colors against their saturation shown in the game. Therefore, these curves were modified in order to show only specific values from the color spectrum, including a range of yellow, blue, and red colors. Thanks to this, only the electricity and water and potentially fire are colored in the game, fomenting gameplay with higher contrasts. This is shown in Fig. 6.54.

Fig. 6.53 shows the difference between enabling the `grading curves` and disabling them.

It has to be noted that to make this effect have the desired impact, all the materials from all the other elements had to be green so the `grading curves` could filter them away, leaving them black and white. The reason behind making them green and then filtering the colors away instead of making them black directly is that the second option makes the elements lose their tonalities.


(a) `Grading curves` off.

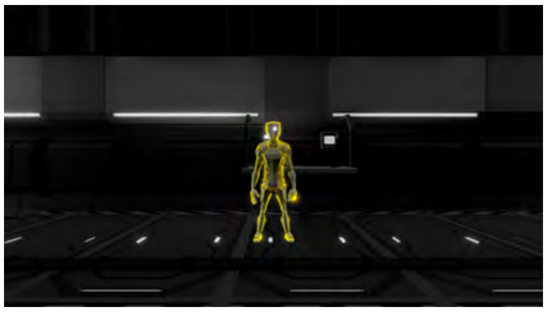

(b) `Grading curves` on.

Figure 6.53: Influence of the `grading curves` from the `color grading` post-processing effect.

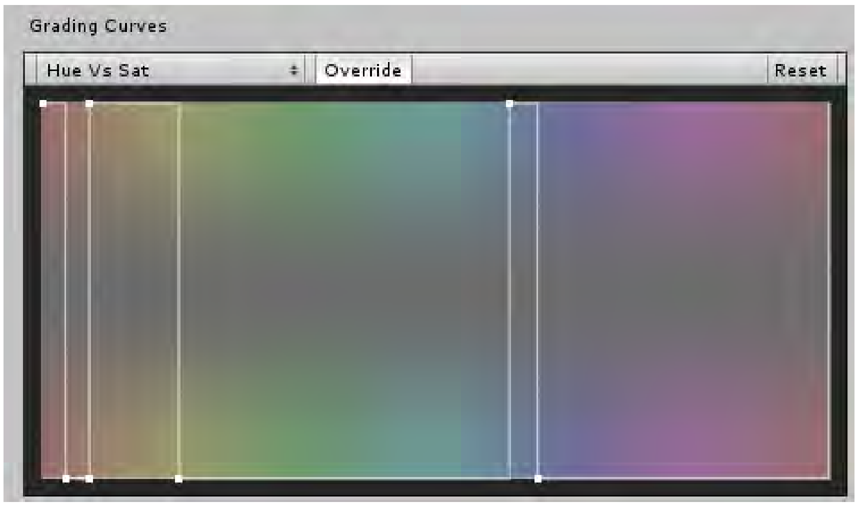
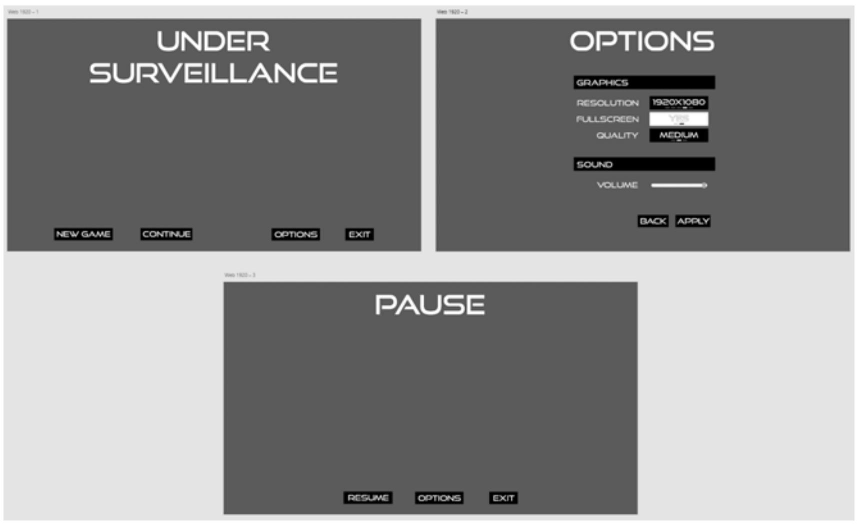


Figure 6.54: Values of the `grading curves` inside of the `color grading` post-processing effect.

6.6.8 Menus and UI

As opposed to the previous versions, the final version of the game features almost no user interface at all. The initially planned dialogue system was discarded, and the authors tried to convey all the feedback and information by the environment elements themselves. Nonetheless, menus were still required to have a complete product. In particular, the main menu, a pause menu and an options menu were implemented.

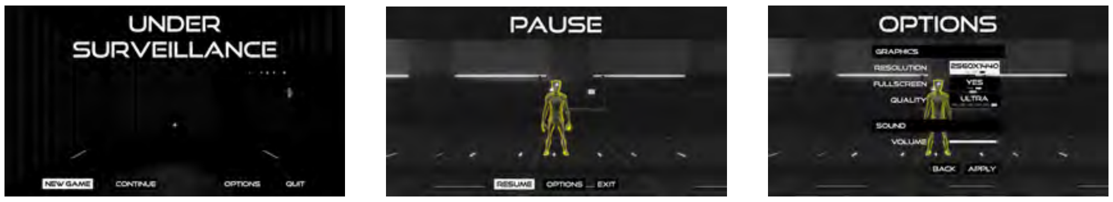
The menus were designed at a late stage of the development of the game since they do not affect the gameplay. Nonetheless, its design also went through the basic stages required when creating any interaction design-related artifact. In particular, before implementing the menu in the game itself, it was prototyped using *Adobe XD*. Fig. 6.55 shows the high fidelity prototype made using this software. The font used for the menu is *Good Times*.



RESOLUTION 1920X1080
FULLSCREEN YES
QUALITY MEDIUM

Figure 6.55: Menu prototype in *Adobe XD*.

Fig. 6.56 shows the previously prototyped menus implemented in the final game. As can be seen, the high fidelity nature of the prototype makes it very similar to the implemented version.



RESOLUTION 2560X1440
FULLSCREEN YES
QUALITY ULTRA

(a) Main menu. (b) Options menu. (c) Pause menu.

Figure 6.56: Menus of *Under Surveillance*.

In relation to the user interface design principles introduced in Sec. 4.4, pliancy has been provided to all the buttons of this user interface. Static hinting is conveyed by having the buttons designed so they are squared shaped with a flat-colored background. This is a fairly common way to design a button according to many design guidelines, such as *Material Design* [67]. Dynamic hinting is achieved by making the button background turn white when the mouse moves over it. Pliant response hinting has also been implemented by making the button background change to a darker color when the mouse is clicked but not released. Finally, cursor hinting was not implemented because it was observed that in desktop applications regular buttons do not have this type of behavior.

6.6.8.1 Cursor

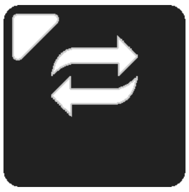
Concerning the parts of the user interface that are not menus, it is important to point out the mouse cursor behavior during gameplay. Since using the mouse is an essential part *Under Surveillance*'s interaction, authors realized that having a custom cursor icon would help the player understand that the game is actively recognizing the presence of a cursor, and thus it is relevant for the gameplay.

Furthermore, it was also decided that the cursor icon could be also used to provide feedback by implementing cursor hinting on the environment elements of the level. Consequently, several cursor icons that convey the action that will occur when the user performs a click, were included in the game. Fig. 6.57 shows five of these cursors.

(a) Throw electricity cursor.

(b) Interactive device cursor.

(c) Throw water cursor.


(d) Swap ability cursor.

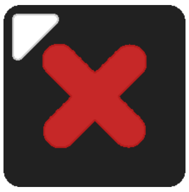

(e) No possible action cursor.

Figure 6.57: Five different in-game cursors.

The script `ProtagonistCursorController` controls which cursor should be displayed at any moment.

6.6.9 Sound Design

Supporting the visual style and narrative design from *Under Surveillance* with an appropriate work of sound design was a must.

All the sounds are taken from the *Retro Future Music Pack* and *Sci-Fi Sound Pack* purchased in the *Unity Asset Store*.

Before proceeding any further it is important to understand the most important parameters which have been modified from each sound effect. These parameters are:

- `Play On Awake`: If activated, the sound is reproduced when the game object that holds the sound effect is instantiated.
- `Loop`: If enabled, the sound is reproduced an infinite number of times.
- `Spatial Blend`: Select if the sound is 2D or 3D. If 3D, the sound is affected by the spatial position and spread. If 2D, all spatial attenuation is ignored.
- `3D Sound Settings`:
  - `Doppler Level`: How much the pitch of the sound is changed based on the relative velocity between the `AudioListener` and the sound.
  - `Max Distance`: The maximum distance the sound stops attenuating at.

To conclude, when 3D sounds are implemented, they are always listened from the perspective of the `AudioListener`, which in this case, it was implemented into the player's avatar.

6.6.9.1 Background Music

Regarding the background music of the game, there is no continuous reproduction of any song, unlike many video games. However, several electronic devices were placed through the world in order to reproduce the background music more organically.

An example of these devices is radio cassettes. These radios are reproducing the looped version of the song *Other World*, part of the *Retro Future Music Pack* when players are in the middle of a puzzle. One of these radios is shown in Fig. 6.58.

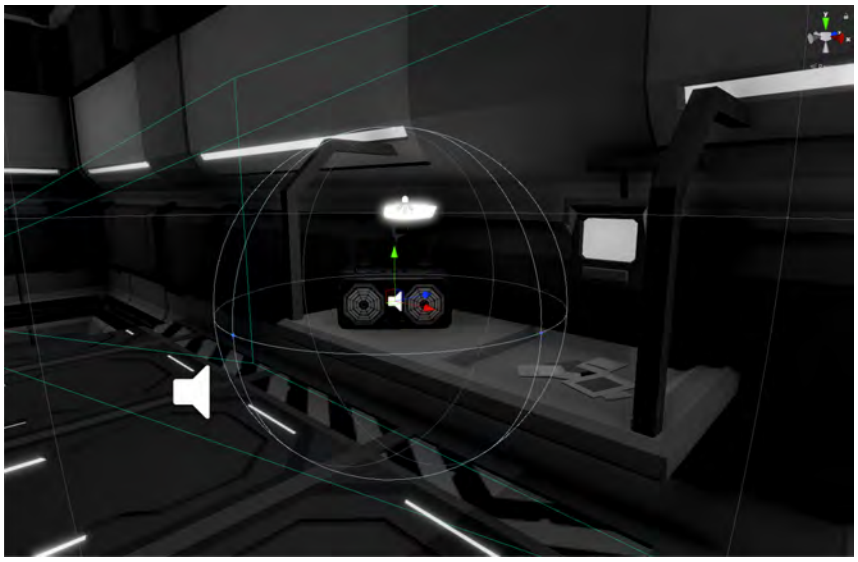
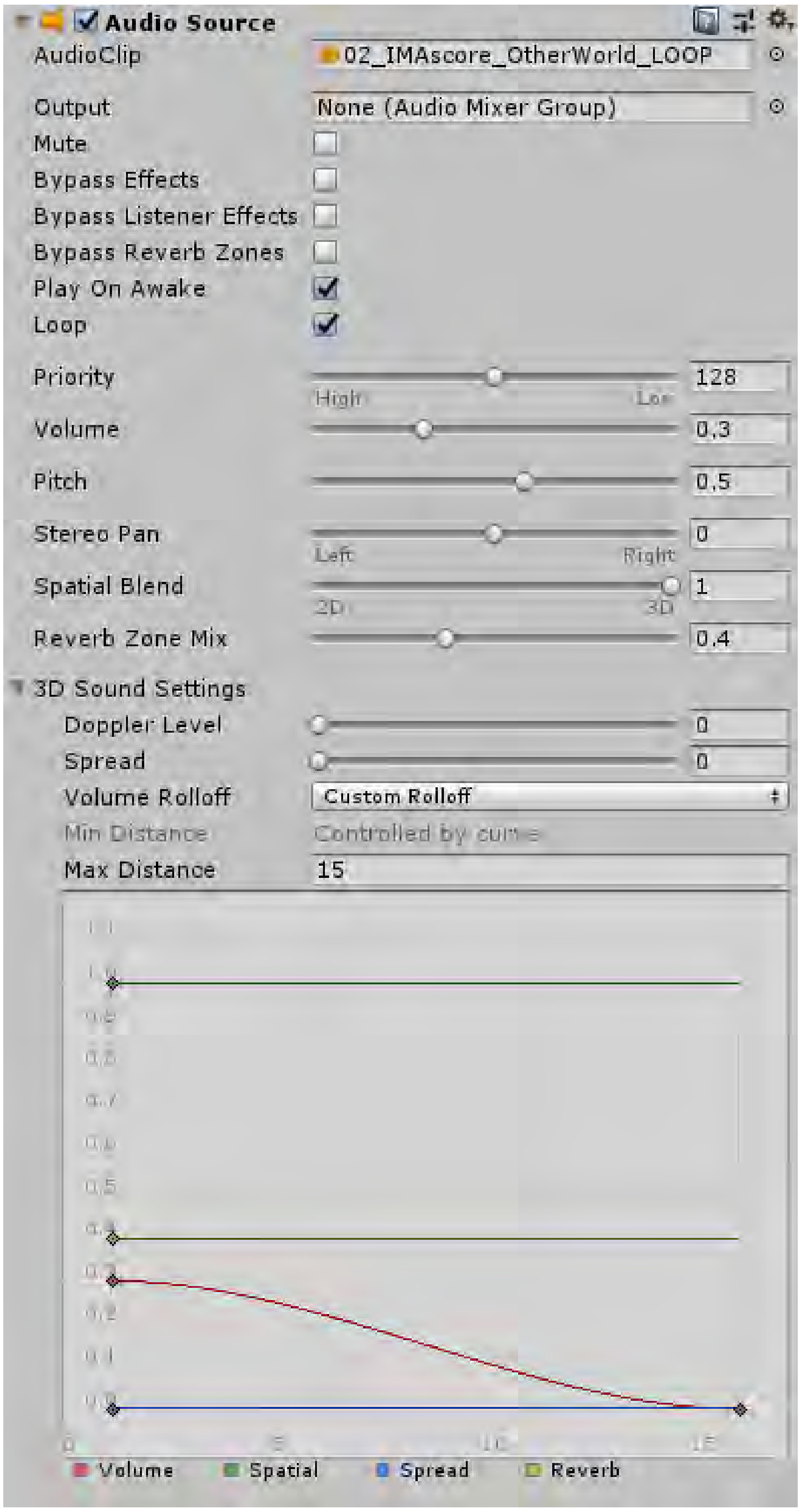


Figure 6.58: A radio placed on a desk next to a puzzle.

To achieve the aforementioned effect so it would blend with *Under Surveillance*'s atmosphere, the spatial blending of the sound was set to 3D, as well as its maximum distance was set to 40 units, so the song is not heard from unintended places. More information about the parameters is shown in Fig. 6.59.



Audio Source
AudioClip: 02_IMAscore_OtherWorld_LOOP
Output: None (Audio Mixer Group)
Mute: ☐
Bypass Effects: ☐
Bypass Listener Effects: ☐
Bypass Reverb Zones: ☐
Play On Awake: ✓
Loop: ✓
Priority: 128 (High – Low)
Volume: 0,3
Pitch: 0,5
Stereo Pan: 0 (Left – Right)
Spatial Blend: 1 (2D – 3D)
Reverb Zone Mix: 0,4
3D Sound Settings
Doppler Level: 0
Spread: 0
Volume Rolloff: Custom Rolloff
Min Distance: Controlled by curve
Max Distance: 15

Figure 6.59: Values of the radio's background music.

Furthermore, if players throw electricity right to the radio, this one will suffer a short circuit momentarily altering the functionality of the radio and deforming the reproduction of the song.

6.6.9.2 Sound Effects

A great number of sound effects were added to complement the background music and to support both atmosphere and gameplay.

Sound effects with the spatial blending assigned to 2D have been used for sounds that are typically not heard within the environment of the game. This includes all the sound effects from menus and HUD.

On the other hand, sound effects with the spatial blending assigned to 3D, do form part of the environment, such as the noise electricity emits or the sound the water makes when thrown.

Although not every single sound effect is listed, the most relevant one towards supporting the atmosphere and gameplay is presented bellow.

Each puzzle ending is marked by a pipe emitting light particles and a font of light the player needs to get in. This pipe is always located at the top-right of the screen. In conjunction with this, a looped sound effect emulating the exterior by a gentle breeze is heard.

The 3D sound is placed on top of the pipe as shown in Fig. 6.60. As for the parameters in detail of this sound effect, achieving the desired effect, are shown in Fig. 6.61.

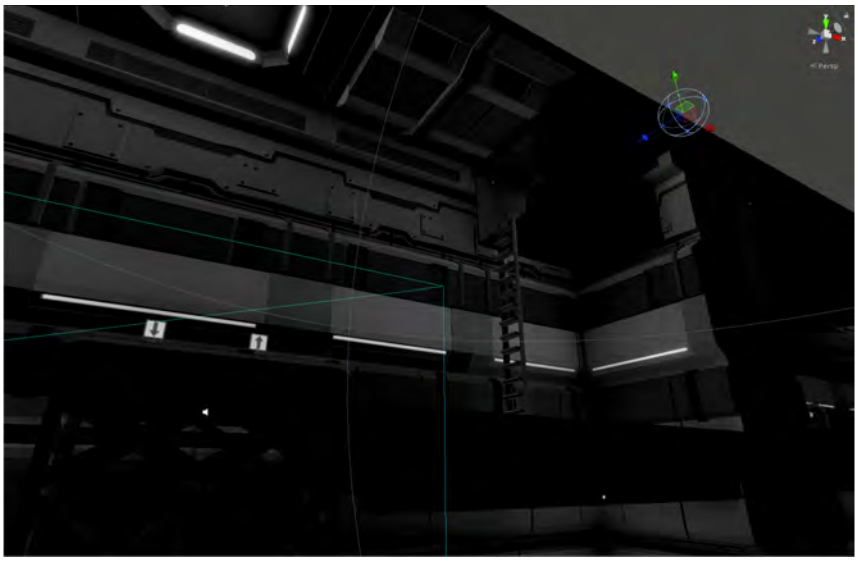
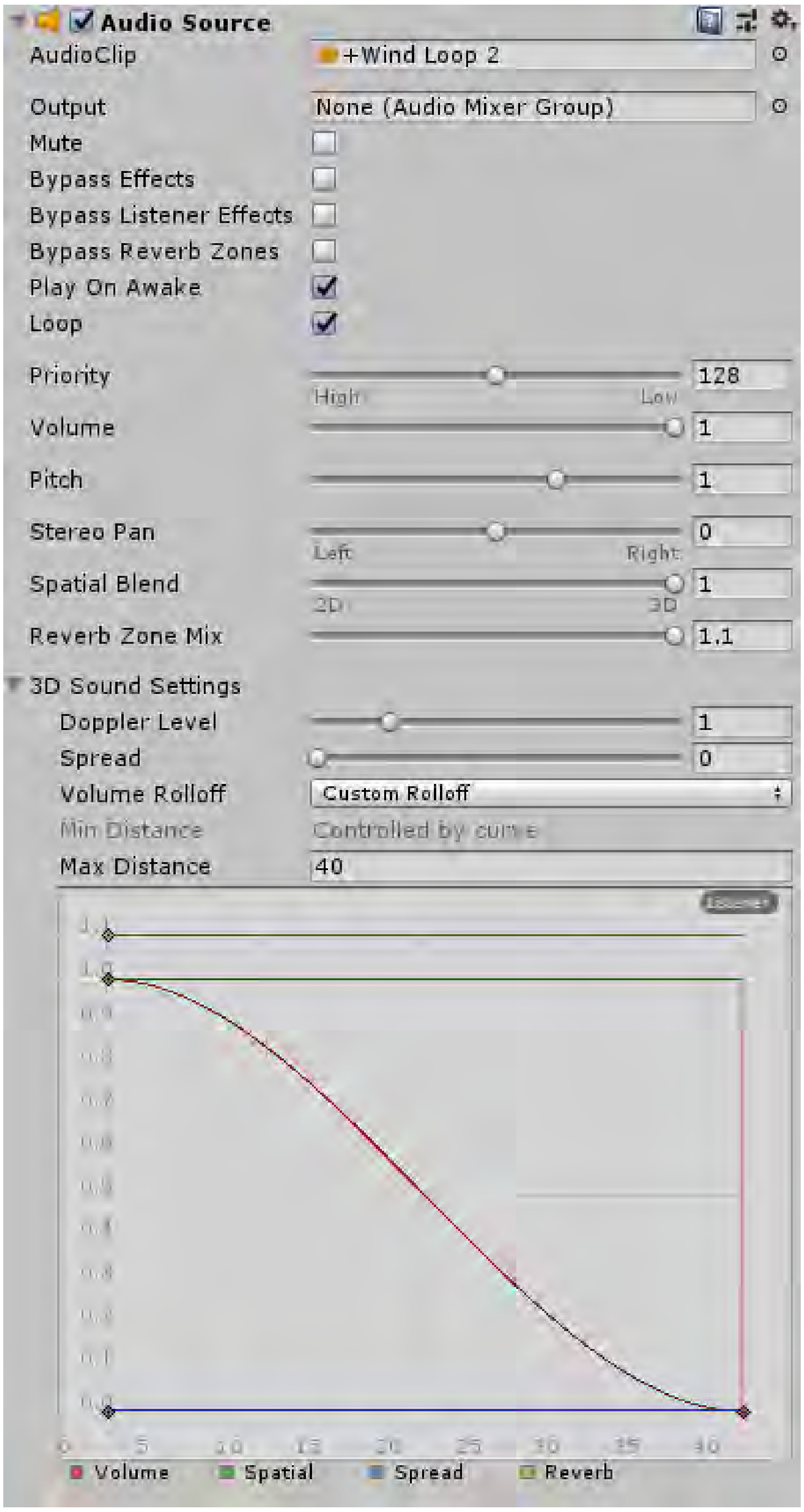


Figure 6.60: Breeze sound effect placed at the top of the pipe marking the end of a puzzle.

This is an example of how to drive the player to a specific position by the usage of sound effects together with other game elements, such as lighting.



Audio Source

AudioClip	+Wind Loop 2
Output	None (Audio Mixer Group)
Mute	☐
Bypass Effects	☐
Bypass Listener Effects	☐
Bypass Reverb Zones	☐
Play On Awake	✓
Loop	✓
Priority	128
Volume	1
Pitch	1
Stereo Pan	0
Spatial Blend	1
Reverb Zone Mix	1.1

3D Sound Settings

Doppler Level	1
Spread	0
Volume Rolloff	Custom Rolloff
Min Distance	Controlled by curve
Max Distance	40

Figure 6.61: Values of the breeze sound effect.

6.6.9.3 Dialogues and Voice Lines

Given the time planned for *Under Surveillance*'s development, it was impossible to hire voice actors to add dialogues and voice lines to the game.

6.7 Phase 7: Analysis

In this phase, the formal process described and explained in Sec. 6.2.2.2 was applied to two different physics-based games. First, the analysis process was applied to *Angry Birds* in order to ensure the effectiveness of the process. Secondly, the same process is applied to *Under Surveillance* to analyze the results of the development and check if they correspond to the initial design described in Sec. 6.3.

6.7.1 Angry Birds

Angry Birds is a 2D puzzle physics-based mobile video game created by *Rovio Entertainment* and originally published in 2009. In *Angry Birds*, the *player* makes use of a slingshot to shoot different birds to multiple fortresses to make them collapse while neutralizing the pigs located at them. A screenshot of the game is shown in Fig. 6.62.

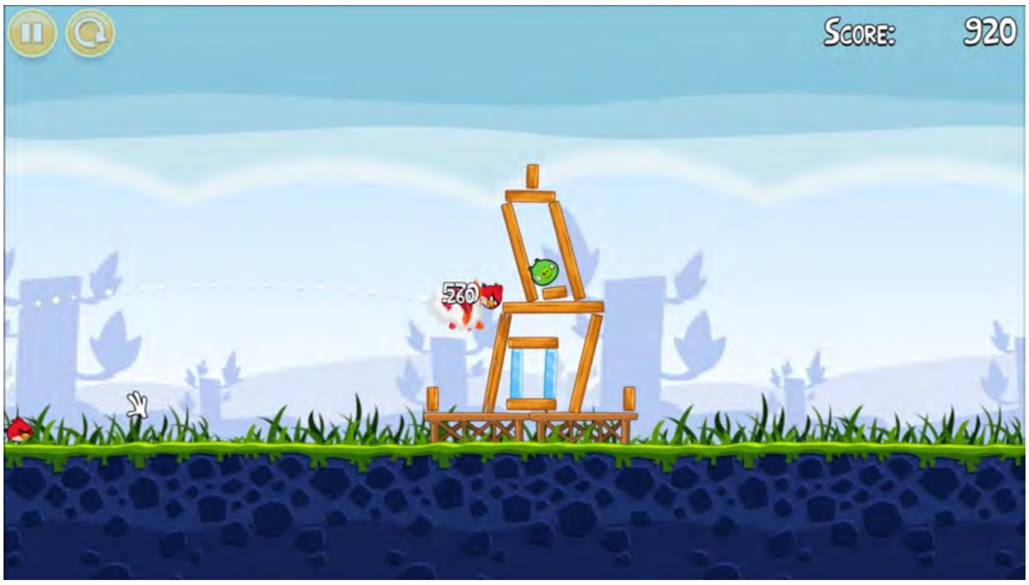


Figure 6.62: Angry Birds in-game screenshot: A bird has just been shot to a fortress. Source: [68]

In order to analyze *Angry Birds*, the PEP framework analysis process has been applied by following all the steps explained in Sec. 6.2.2.2:

6.7.1.1 Identify the Phenomena, the Environment and the Mechanics

A list with all the *phenomena*, the *environment* elements and the *mechanics* has been tailored:

- **Mechanics**: Shoot birds, use the birds' special abilities.
- **Environment**: Earth-like world, planks made of glass, wood and stone, TNT boxes, pigs and birds.
- **Phenomena**: Parabolic trajectories, collisions, damage, destruction and explosions.

6.7.1.2 Validate the Phenomena and the Environment

All the aforementioned identified *phenomena* are then validated by checking whether they are supported by the *environment* elements or not:

- **Parabolic trajectories**: All the objects present in the *environment* behave consistently. They always describe trajectories governed by the same laws of *physics*.
- **Collisions**: All the objects present in the *environment* collide with each other in a consistent manner according to the game's formalized *physics*.
- **Damage and destruction**: Planks consistently take damage when they receive an impact by other objects of the *environment* and they are destroyed once they have received enough damage. Under the same conditions and given the same impact, a plank always receives the same amount of damage. In order to provide a bigger variety of this kind of *phenomena*, planks are made of different materials, differing in the maximum amount of damage they can take before they are destroyed.
- **Explosions**: TNT boxes explode when they receive an impact by a different object with a force greater than a specific and constant threshold. The damage of these explosions to the surrounding objects is always the same, according to the distances to them.

6.7.1.3 Validate the Mechanics

All the previously identified *mechanics* are validated by checking whether they directly or indirectly support the *phenomena*:

- **Shoot bird**: Shooting birds is the main game *mechanic*. By shooting a bird, the *player* can make all the aforementioned identified *phenomena* emerge directly: parabolic trajectories, collisions, damage and destruction when birds hit fortresses and explosions when birds hit a TNT box. Also, these *phenomena* can emerge indirectly due to the chain reaction produced when fortresses break and fall over other *environment* elements. Thus, this *mechanic* supports the *phenomena*.
- **Use bird's special ability**: All the birds' special abilities are designed to increase the potential amount of damage dealt to fortresses. Therefore, this *mechanic* supports the *phenomena*.

6.7.1.4 Evaluate the Deductions

Angry Birds is designed to be an intuitive game. This is achieved both by making use of as much a priori knowledge as possible and by designing levels so new *phenomena*, *environment* elements, and game *mechanics* are steadily introduced.

The following design decisions to support the use of a priori knowledge have been found out when performing this analysis:

- **Earth-like world**: Many *phenomena* simulated in the game behave as they do in the real world. Daily, parabolic trajectories are witnessed in the real world due to the action of gravity. By creating an *environment* reassembling the Earth, the *player* expects to take into account the gravity force before shooting a bird, even if they have never played *Angry Birds* before.
- **Birds**: Since *Angry Birds* is entirely built around the idea of shooting projectiles following parabolic trajectories, these projectiles spend most of their time in mid-air. Naturally, these projectiles became an element present on Earth's nature which is usually found flying around: birds.
- **Slingshot**: Having birds as the game's projectiles to describe parabolic trajectories is not intuitive enough to the *player* as birds do not fly like that. However, when a slingshot is added as part of the *environment* when shooting a bird, the *player* can then safely assume all birds shot with such slingshot will follow a parabolic trajectory, even if they have never interacted with the game's slingshot before.
- **Planks' materials**: Glass, wood and stone are familiar materials and so are their general properties. Furthermore, they are easy to distinguish from each other. This is part of *Angry Birds*' *environment* design, with glass planks that are easily destroyed, stone planks that are hard to break, and wood planks which resilience is somewhere in between the other two.
- **TNT boxes as explosion triggers**: The relationship between TNT and explosions is very common in games as well as it is in the real world. Thus, when the *player* identifies a TNT box in the *environment*, they expect to be able to destroy it and cause a big explosion *phenomenon*. *Angry Birds* is not an exception to this.

On the other hand, there are also aspects of *Angry Birds* which the *player* needs to learn by playing, as their a priori knowledge is not enough. Nonetheless, these aspects are carefully and gradually introduced to the *player*. Thus, the amount of *deductions* made during gameplay is limited, not overwhelming the *player* with a burst of new concepts early in the game. Thanks to this design approach, a posteriori knowledge can more easily be built by the *player* around these concepts.

One of the most relevant and illustrative examples of how these concepts are introduced is how new birds are presented to the *player*. First, when a new type of bird is introduced, it is always done once the *player* has completed a fair amount of levels in which no new bird has appeared. Secondly, in the levels where a new type of bird appears, it is always done by following a specific structure. This structure consists on letting the *player* shoot several instances of the same new type of bird and none of the others. This improves the quality of the *deductions* the *player* makes when experimenting with a new element from the *environment*, such as the new bird, by letting the *player* trigger more instances of the same *phenomenon*, leading to a more accurate a posteriori knowledge.

6.7.2 Under Surveillance

With the final version of *Under Surveillance* already developed, it is necessary to validate the result. In order to carry out this validation, an analysis using the PEP framework methodology was performed.

As hinted previously, some of the elements that were originally part of the design of the game, were not ultimately implemented due to time restrictions. In particular, the fire *phenomenon* and its related *environment* elements and *mechanics* are not part of the final product. In consequence, the subsequent sections provide an analysis of the final version of the game as it is implemented, with electricity and water.

6.7.2.1 Identify the Phenomena, the Environment and the Mechanics

A list with all the *phenomena*, the *environment* elements and the *mechanics* present in the final version of *Under Surveillance* has been tailored:

- **Mechanics**: Throw electricity, throw water.
- **Phenomena**: Electricity propagation through conductive materials, electricity interaction with electrical devices, water movement according to fluid dynamics and water buoyancy.
- **Environment**: Metallic and non-metallic objects which conduct and isolate electricity, respectively, electrical devices that can be interacted with electricity, swimming pools, objects that float.

6.7.2.2 Validate the Phenomena and the Environment

All the aforementioned *phenomena* are then validated by checking whether they are supported by the *environment* elements or not:

- **Electricity propagation through conductive materials**: All the objects present in the *environment* fall in one category, either they are metallic or they are non-metallic. Metallic materials always react by propagating electricity, which is visually shown by generating electric sparks and arcs. Non-metallic materials do not react to electricity.
- **Electricity interaction with electrical devices**: A series of visually identifiable electrical devices are placed throughout the *environment.* These devices consistently react to electricity when thrown by the player. The reaction can be always witnessed in real-time by the player.

- **Water movement according to fluid dynamics**: All the concave areas (swimming pools) present in the environment can be filled with water, which will inside accumulate them.
- **Water buoyancy**: All the wooden-like objects present in the *environment* float when they are on the water.

6.7.2.3 Validate the Mechanics

All the previously identified *mechanics* are validated by checking whether they directly or indirectly support the *phenomena*:

- **Throw electricity**: The player can throw electricity to any object present in the *environment.* If the player throws electricity to a metallic object, the *phenomena* "electricity propagation through conductive materials" emerges. On the other hand, if the player throws it to to an electrical device, the *phenomena* "electricity interaction with electrical devices" emerges. Therefore, this *mechanic* supports the *phenomena.*
- **Throw water**: The player can throw water anywhere in the *environment.* If the player throws the water inside a concave area, water will accumulate, raising its level the more water the player throws. Therefore, by doing this the *phenomena* "water movement according to fluid dynamics" emerges. At the same time, if any wooden-like object is present in this area, it will float as the water level rises, which implies the emergence of the *phenomena* "water buoyancy".

6.7.2.4 Evaluate the Deductions

When designing *Under Surveillance*, both authors agreed on having a strong focus on making the game as intuitive as possible. To ease the learning curve of the game, the design heavily relies on players' a priori knowledge. On top of that, the main two *mechanics* are not present since the beginning of the game. Instead, the player can initially only throw electricity. When they make a fair amount of progress and are usually familiar with this mechanic, the throw water mechanic is introduced.

The following design decisions to support the use of a priori knowledge can be identified in the game:

- **Real-world inspired objects**: Most of the *environment* elements of the game are representations of objects that exist in real life and which material is identifiable. For instance, jails are known to be metallic. This helps the player easily identify whether an object is metallic or not, and, consequently, whether it will conduct electricity or not.
- **Real-world inspired phenomena**: The behaviors of both electricity and water were not at all created from scratch by the designers. Instead, they are representations of their behaviors in the real world. This helps the *player* to predict how the *environment* will react to both electricity and water even without having used it yet.

On the other hand, there are also aspects of *Under Surveillance* that the *player* needs to learn by playing, as their a priori knowledge is not enough. For this reason, authors have designed the game so that these concepts are gradually introduced, in an attempt to not overwhelm the *player* with a large number of new ideas to be learned in a short period.

The clearest example of this design philosophy is the fact that the main character's abilities are not presented simultaneously. Instead, the main character is provided with electricity powers first. It is only after the player has made enough progress through the game and is therefore familiar with this ability, that water is introduced.

7

Results

7.1 The PEP Framework

The first major outcome of the thesis is the formal method created by the authors to develop physics-based games, namely the PEP framework (standing for *Player*, *Environment* and *Physics*). It is important to mention that the method has been already thoroughly explained in Sec. 6.2 and applied in Sec. 6.3 and Sec. 6.7. Consequently, this section provides only an overview of the framework.

Fig. 7.1 shows the architecture of the framework.

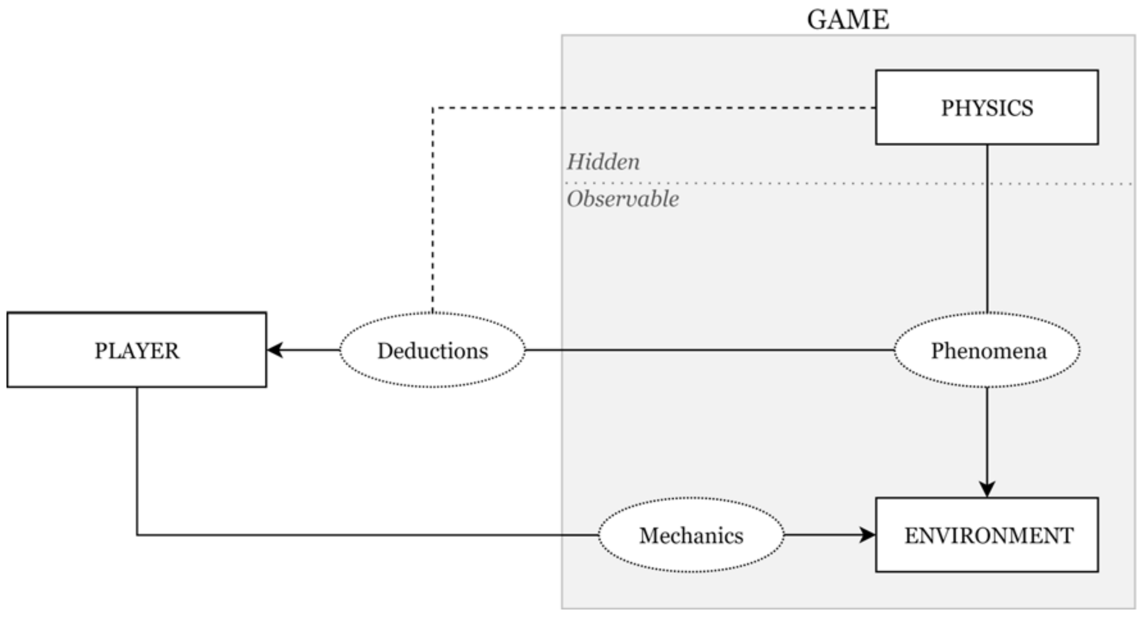


Figure 7.1: The PEP framework architecture.

According to the PEP framework, physics-based games can be broken down into three main components and three secondary components that emerge from the interactions among the former.

The three main components are:

- **Physics**: Set of rules governing the *environment*.
- **Environment**: Materialization of the current game state [10].

- **Player**: External user who interacts with the *environment*.

The three secondary components are:

- **Mechanics**: Methods invoked by the *player*, designed for interaction with the *environment* [61].
- **Phenomena**: Observable events or occurrences manifested in the *environment*, consequence of the *physics*.
- **Deductions**: *Player*'s knowledge gain regarding how the *physics* work derived from observed *phenomena*.

7.1.1 Applying the framework

The PEP framework can be applied both to design and analyze physics-based games. This section provides an overview of both processes, of which the complete explanation can be found in 6.2.2.

The design process consists of the following steps:

1. **Design phenomena**: Game designers propose *phenomena* which will be present in their new physics-based game.
2. **Design environment**: Taking as a starting point the *phenomena* designed in the previous step, an *environment* is derived from them.
3. **Formalize physics**: Once the *phenomena* and the *environment* have been defined, the *phenomena* behaviors in such *environment* are generalized by defining laws of *physics*. These set of laws will rule all the *environment* and all its emerging *phenomena*.
4. **Design mechanics**: How the *player* will interact with the *environment* must be specified. These *mechanics* must support the *phenomena* by allowing the *player* to directly or indirectly originate, alter or bring to an end *phenomena*.
5. **Design deductions**: Designers must ensure the game's *physics* are intuitive to the *player*, by limiting the amount of *deductions* the *player* needs to make during gameplay.

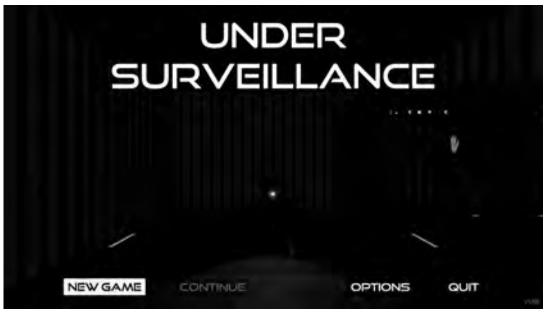
The analysis process consists of the following steps:

1. **Identify the phenomena, the environment and the mechanics**: All the *phenomena*, the *environment* elements and the *mechanics* are listed.
2. **Validate the phenomena and the environment**: The *phenomena* identified in the previous step must be validated by checking whether or not the *environment* supports such *phenomena*.
3. **Validate the mechanics**: The *mechanics* identified in the first step must be validated by checking if they directly or indirectly support the *phenomena*.
4. **Evaluate the deductions**: The way the player learns about the game's physics must be analyzed. Every *phenomenon* which cannot be explained by

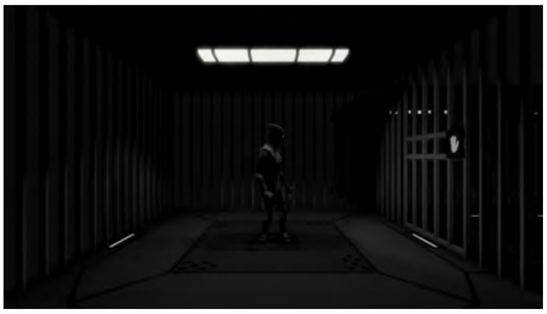
the *player*'s a priori knowledge will lead them to make *deductions*, establishing a posteriori knowledge about how the game's *physics* work. The more gradually the *player* needs to make *deductions*, the less overwhelming the game becomes when learning about it and its *physics*.

7.2 Under Surveillance

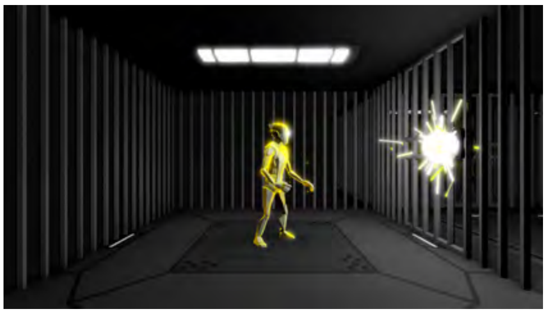
After developing *Under Surveillance* throughout all the phases explained in Sec. 6, a finalized and playable version of the game was obtained. In Fig. 7.2 multiple screenshots of the final version of *Under Surveillance* are shown.



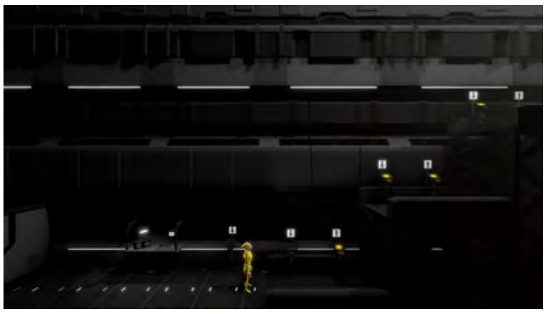
(a) Main menu.



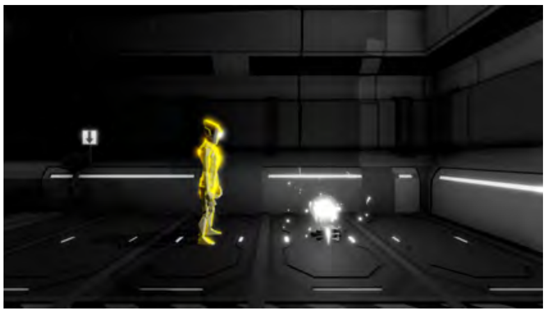
(b) The robot is trapped in a jail cell.



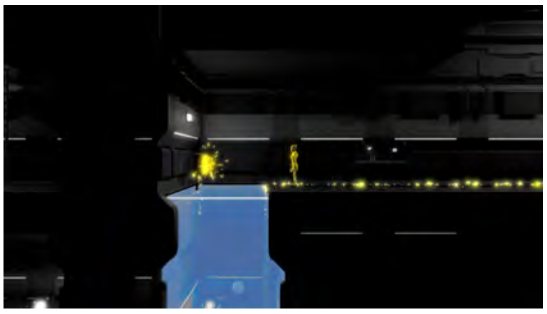
(c) The robot throws electricity to open the door of the jail cell.



(d) The robot encounters the first puzzle room.



(e) The robot discovers a secret collectible hidden in the first puzzle room.



(f) The robot fills a pool area with water in order to restore the electricity of the second puzzle room.

Figure 7.2: Final version of *Under Surveillance*.

The rest of this section presents an overview of this result, as well as how it was distributed and promoted.

7.2.1 Gameplay

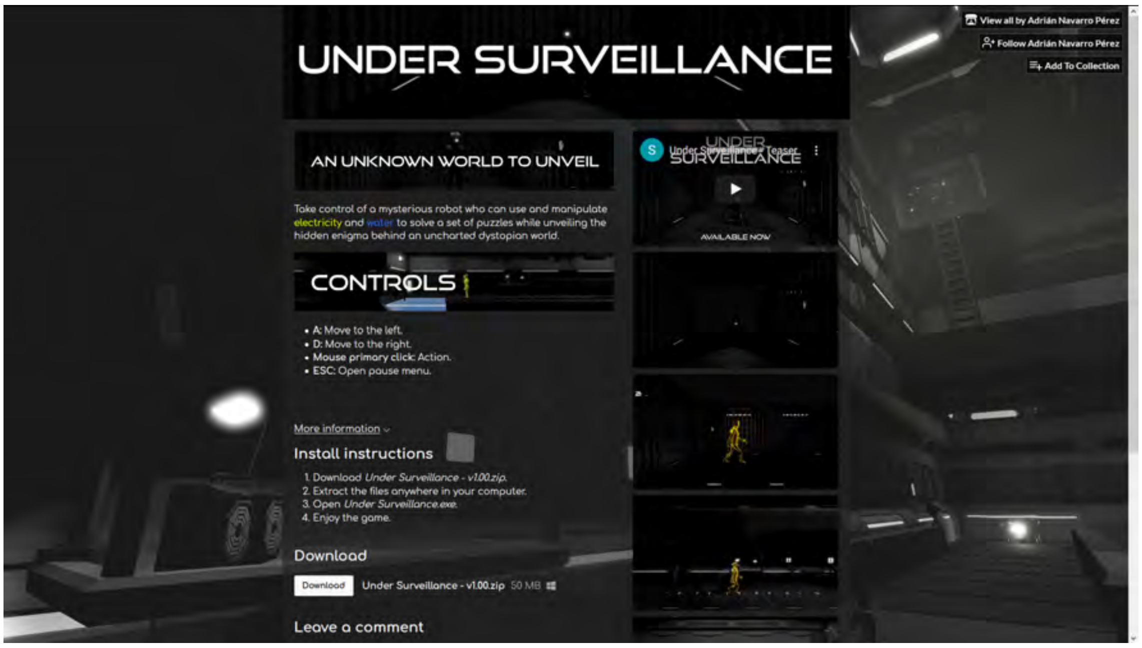
Since the developed product is a game, the best way to showcase all its features and its potential is through a gameplay video. The following *YouTube* link contains a complete gameplay of *Under Surveillance*: `https://youtu.be/ix5ictxuMVQ` [69].

7.2.2 Distribution

As stated in Sec. 5.1.1, the master thesis authors were interested in developing a game, not only to answer the research question presented in Sec. 1.2, but also "to increase the extension and quality of their portfolio". Publishing and distributing the game was essential to achieve this.

The platform chosen to publish the game, due to its popularity, its features and its affordability was *Itch.io* [70]. *Itch.io* provides free storage for the compiled game as well as any other resources related to the game. It also allows for the creation of a personalized web site that potential players can visit to obtain information about the game and download it. This feature allows to drastically increase the public visibility of the game.

Fig. 7.3 shows the final look of the web page created for *Under Surveillance* using *Itch.io*. The link to that page is: `https://adriannp57.itch.io/under-surveillance` [71].


View all by Adrián Navarro Pérez
Follow Adrián Navarro Pérez
Add To Collection
UNDER SURVEILLANCE
Under Surveillance - Teaser
AVAILABLE NOW

UNDER SURVEILLANCE

AN UNKNOWN WORLD TO UNVEIL

Take control of a mysterious robot who can use and manipulate electricity and water to solve a set of puzzles while unveiling the hidden enigma behind an uncharted dystopian world.

CONTROLS

- A: Move to the left.
- D: Move to the right.
- Mouse primary click: Action.
- ESC: Open pause menu.

More information

Install instructions

1. Download *Under Surveillance - v1.00.zip*.
2. Extract the files anywhere in your computer.
3. Open *Under Surveillance.exe*.
4. Enjoy the game.

Download

Download Under Surveillance - v1.00.zip 50 MB

Leave a comment

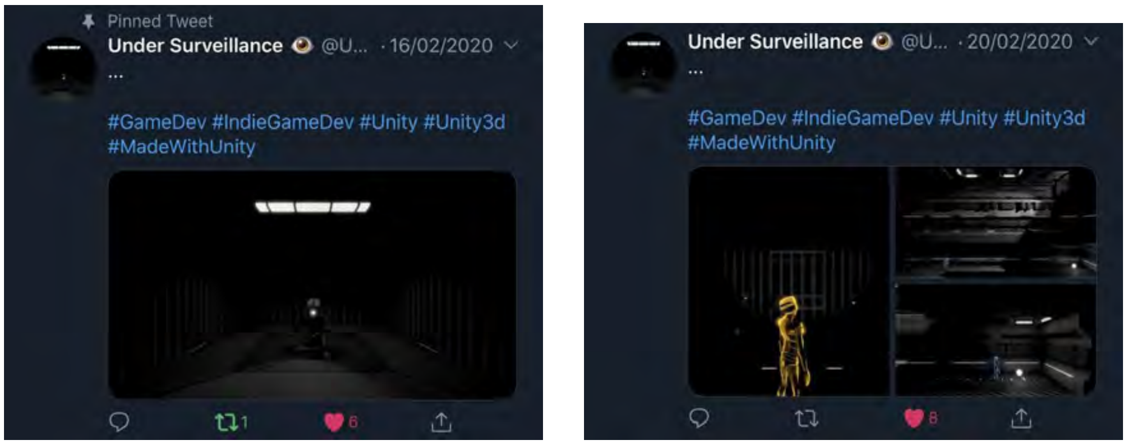
Figure 7.3: *Itch.io* web page of *Under Surveillance*.

7.2.3 Social Media

In order to further increase the visibility of the game, reaching more potential players, social media accounts were created to promote *Under Surveillance*. In these accounts, the authors posted relevant content related to the state of the development of the game. The goal behind this was to build a small community before releasing the game. This would help to increase its popularity during the first days of its release, which are critical for a game to succeed.

Fig. 7.4 and Fig. 7.5 show examples of posts and promotional content published in social media.

Figure 7.4: Example of one of the multiple content used to promote *Under Surveillance.*



Pinned Tweet
Under Surveillance @U... · 16/02/2020
...
#GameDev #IndieGameDev #Unity #Unity3d #MadeWithUnity
1 6

Under Surveillance @U... · 20/02/2020
...
#GameDev #IndieGameDev #Unity #Unity3d #MadeWithUnity
8

Figure 7.5: Posts published on the social media accounts of *Under Surveillance.*

8

Limitations, Discussion and Ethical Considerations

8.1 Limitations and Challenges

Some limitations must be taken into account when making use of the PEP framework and when playing *Under Surveillance*. This section covers the most relevant limitations of both.

8.1.1 The PEP Framework

First, as mentioned in Sec. 1.4, this work does not intend to become the only method to apply when designing or analyzing physics-based games. The framework does not aim to help to face all the challenges that need to be tackled when creating a game or analyzing it. For example, the PEP framework does not take into consideration the *player*'s desirable emotional responses (aesthetics) during gameplay, which is addressed by other formal approaches such as [15].

Secondly, extensive testing of the PEP framework has not been conducted. Consequently, how well the framework performs at edge cases is yet to be demonstrated.

Lastly, as for the generation of *deductions* during gameplay, it has been assumed the *player* has no previous experience with the game. This might not be the case in some scenarios in which the *player* has watched other people playing the same game or they have gathered information about the game in any other way.

8.1.2 Under Surveillance

As for Under Surveillance, it is important to recall that the initial scope was reduced a few times. In the first paper prototypes shown in Sec. 6.1.1 the game was planned to contain a considerable amount of natural elements and interactions. Most of these were discarded before the first formal design of the game, presented in Sec. 6.3. Furthermore, during the implementation phases, presented in Sec. 6.4, Sec. 6.5 and Sec. 6.6, the scope was again reduced by eliminating the fire and all its related interactions from the game.

To summarize, the authors had an initial scope that was too ambitious considering the amount of time given to develop the master thesis. Nonetheless, since the agile practices explained in Sec. 4.1.1 were applied, it was possible to dynamically solve this problem, by eliminating or modifying some of the requirements of the game without creating a significant overhead in terms of workload.

8.2 Discussion

As mentioned in Sec. 5.2.1, the authors planned to submit the work related to the PEP Framework to a relevant conference to obtain feedback from experts on the field and potentially have a published article.

After considering several options, the work was ultimately submitted to the *IEEE Conference on Games (CoG) 2020.* [72]. A scientific paper was written, collecting all the relevant information regarding the PEP Framework. This paper is attached in Sec. A.

In relation to the outcome of the submission, it is important to first understand how the peer review process works. Each work is reviewed by four anonymous experts on the field, each of whom grades the submission by giving it a numeric score between -3 and 3. A negative score means that the paper is rejected, while a positive score means that it is accepted. The greater is the absolute value of the score, the strongest is the rejection or acceptance. A score of zero implies that the paper is a borderline case.

After the PEP Framework paper was reviewed by four experts, the results were emailed to the authors. The following paragraphs present the review results, summarizing the most relevant parts from the comments provided by the reviewers.

Reviewer 1: -1 points.
Given the strong philosophical background, the paper seems more suitable to other conferences than IEEE. Even if the PEP Framework has some value, game developers usually design their games around what game engines offer. It is suggested to either leave the paper as it is and submit to a different conference or that the philosophical discussions are toned down to something more practical.

Reviewer 2: -2 points.
The grounding theories of Plato and Kant are somewhat out-of-context and detract rather than add to the paper. The mental model is more relevant but has already been explored extensively in the game studies literature. Therefore the paper feels shaky in terms of novelty. It is also unclear how it specifically is best-fitted for physics-based games, since the mental model, and player's observations of (often opaque) phenomena are a big part of all games.

Reviewer 3: 0 points.
Even though the paper proposes an interdisciplinary approach to physics-based games, we already have tools to design even more complex scenarios, mechanics and physics. Therefore, the added value is quite limited.

Reviewer 4: 2 points.
The psychology and philosophy backgrounds are adequately described. The authors

present two examples in order to prove the validity of the proposal. My final evaluation of the paper is good, and in my opinion, it can be accepted. Still, no "Related Work" section and no references about physics-based games are provided. I would suggest to review that.

Overall, even though the paper was rejected, one of the reviewers considered it was a borderline case and another considered that it could be accepted. Also, one of the reviewers that rejected the paper advised the authors to try to submit it to a more appropriate conference. Hence, the developed framework and the paper have at least some potential and they may have a strong value after some polishing.

8.3 Ethical Considerations

The video game industry is not free from ethical and societal problems that are confronted every day. As part of the development of *Under Surveillance*, the master thesis authors addressed three of the most relevant, and sometimes controversial, according to their code of ethics, as described in the following subsections.

8.3.1 Gender Equality

From the very early days of the video game industry, it has always faced big ethical and societal problems regarding gender inequality where women have been denied their human rights, both inside and outside the video games themselves. Denying women from full participation in leadership and decision-making roles within the industry is just one of the uncountable obstacles women need to face every day within the field to achieve the same as a man.

Although the industry is slowly advancing towards the end of the discrimination against women and girls, this persists. Therefore, half of the population cannot enjoy participating in the industry as they deserve and they cannot enjoy playing to the games they produce at its fullest. Thus, as part of what the master thesis authors believe, Gender Equality [73], as described in the Sustainable Development Goals from the United Nations [74], was taken into account when designing the narrative design of *Under Surveillance*. The main character is designed so every player can feel identified with it, regardless of their gender, unlike many other video games where their main protagonists are usually a white middle-aged man, fomenting the aforementioned discrimination.

8.3.2 Quality Education

Video games are also a great opportunity to ensure inclusive and quality education while they can be lifelong learning opportunities for all. Video games can make players liberate their intellect and unlock their imagination, which both are fundamental pieces for self-respect.

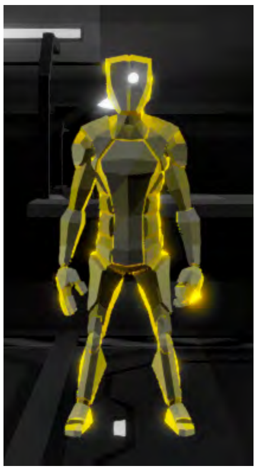
Since Under Surveillance is a puzzle-solving game, a learning experience was carefully designed to hone and enhance the player problem-solving skills. This foments and supports the 4.4 target goal from the Quality Education [75] sustainable development goal: Increase the Number of People with Relevant Skills for Financial Success.

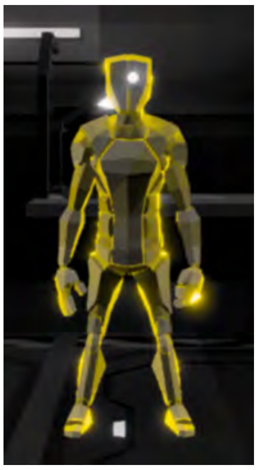
8.3.3 Reduced Inequalities

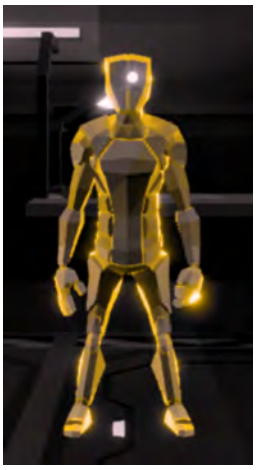
As game developers, the master thesis authors want *Under Surveillance* to be played by the greatest possible number of people, reducing inequalities, as described in Reduced Inequalities [76] from the Sustainable Development Goals. Taking into account accessibility is crucial to achieving this since more people will play *Under Surveillance* if they are physically able to enjoy it.

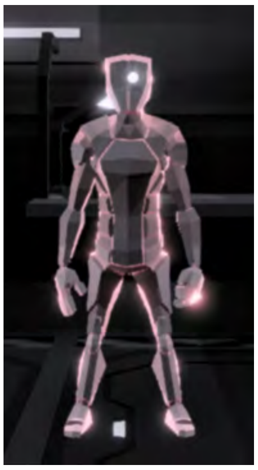
One of the main concerns the authors had during the development of *Under Surveillance* is how relying on color to provide contrast and feedback could affect colorblind players. This was particularly important for the main character, whose current enabled ability is displayed by rendering it with a specific color (yellow for electricity and blue for water). Fig. 8.1 shows both states of the main character with filters that display its colors as perceived by people with different types of color blindness.

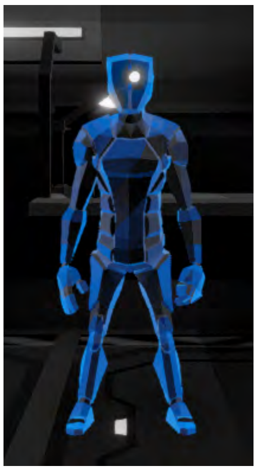
As can be seen, whether the player has protanopia, deuteranopia or tritanopia, blue and yellow can still be distinguished from one another. Therefore, in terms of contrast and differentiation of character states, color blindness does not seem to cause any major problems. Nonetheless, even if these players can distinguish these two colors, it is still pending to study whether they can easily identify electricity as electricity and water as water.

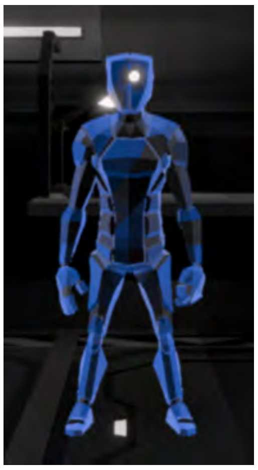

(a) Original.

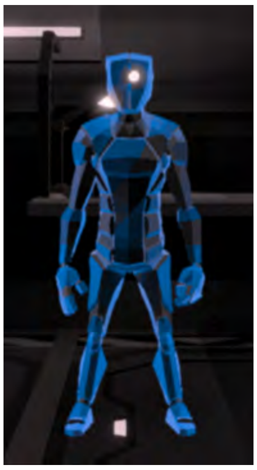

(b) Protanopia.


(c) Deuteranopia.


(d) Tritanopia.


(e) Original.


(f) Protanopia.


(g) Deuteranopia.

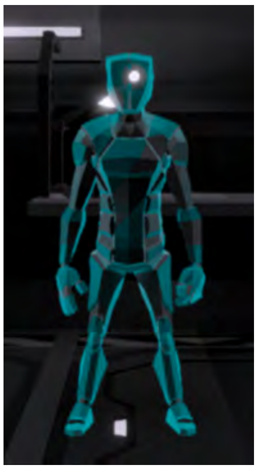

(h) Tritanopia.

Figure 8.1: Main character visualized with different color blindness filters.

9

Conclusion and Further Work

9.1 Conclusion

This master thesis has introduced, explained and applied the PEP Framework, a formal method to design consistent and intuitive physics-based games. Furthermore, the method was applied to develop *Under Surveillance* a 2.5D puzzle-solving adventure video game. The relevant background, theory and methodologies were described, along with a plan on how to tackle the work of the thesis. Then, how the process was executed and its results were presented. Finally, the limitations, challenges and ethical considerations of the developed work were pointed out.

The developed work aimed to answer the research question presented in Sec. 1.2, which reads as follows:

"What should be considered when designing physics-based interactions between player and environment elements to achieve more engaging experiences in physics-based games?"

In the first phase of the execution presented in Sec. 6.1, the design of a physics-based game without taking into consideration the physics-based interactions between player and environment elements was performed. The master thesis authors realized that they were just expressing different ideas on paper prototypes without following any particular process. This resulted in a chaotic outcome rather than a useful game design paper prototype.

Therefore, a formal process along with a vocabulary, namely the PEP framework, were designed in Sec. 6.2. This process conducts the design of these interactions and their emerging events consistently and intuitively to the player. As part of proving the effectivity of such framework, *Under Surveillance* was re-designed in Sec. 6.3 and successfully implemented in Secs. 6.4–6.6. Finally, the PEP framework was used to analyze *Angry Birds* and *Under Surveillance* in 6.7.

Consequently, answering to the question, the considerations that must be taken into account when designing physics-based interactions are those included in the PEP framework methodology, summarized in Sec. 7.1 and deeply explored in Sec. 6.2.

9.2 Further Work

Both the PEP Framework and *Under Surveillance* can be further extended and polished. This section presents possible further work for both of these parts of the thesis.

9.2.1 The PEP Framework

While developing the framework, it was postulated that without changing its core architecture, but only slightly modifying some of its components, it could also be used to design and analyze games that are not necessary physics-based.

As stated by the summary by **Reviewer 2** in Sec. 8.2, "The mental model, and player's observations of (often opaque) phenomena are a big part of all games", which gave more strength to the aforementioned postulation.

Although this aspect of the PEP framework has not thoroughly been explored, the authors realized that most of its components are common to most games. *Environment*, *player* and *mechanics* are present in every game by definition. As for the *physics*, these could be generalized to all the rules governing a game. Hence, the concept of *phenomena* would still be present in any game, not as the emergence of observable occurrences as a consequence of the *physics*, but rather as a consequence of the game rules. Conclusively, the *deductions* would be related to how these rules work.

Nonetheless, games in which the aforementioned aspect of the PEP framework does not apply do exist. When all the game rules are known by the *player* beforehand, which is common in board games, *deductions* are not present since *players* are not necessarily expected to learn the rules during gameplay.

9.2.2 Under Surveillance

One extension that can be applied to *Under Surveillance* is adding fire as the authors envisioned in their original design. Due to time constraints, this element was discarded. Adding it would provide a richer and longer experience. As a consequence, more levels would need to be designed and implemented accordingly.

Furthermore, user testing was not carried out. By having different people to test the game and provide feedback, the product could have been greatly improved. Luckily, once the game was published to *Itch.io*, several users downloaded and played the game. One of them even uploaded a gameplay video to *YouTube* in which it was checked that indeed, some of the deductions designed by applying the PEP framework to the game during its design phase, were successfully carried out. The comment in which they post the gameplay video can be found in the *Itch.io* page of *Under Surveillance* [71].

Bibliography

[1] O. Morgenstern and J. von Neumann, *Theory of Games and Economic Behavior.* Princeton University Press, 1944.

[2] J. Huizinga, *Homo Ludens: A Study of the Play-Element in Culture.* Routledge & Kegan Paul Ltd, 1949.

[3] R. Caillois, *Man, Play, and Games.* Thames & Hudson, 1961.

[4] C. C. Abt, *Serious Games.* Viking Press, 1970.

[5] E. M. Avedon and B. Sutton-Smith, *The Study of Games.* John Wiley & Sons, 1971.

[6] C. Crawford, *The Art of Computer Game Design: Reflections of a Master Game Designer.* Osborne/McGraw-Hill, 1984.

[7] B. Suits, *The Grasshopper: Games, Life and Utopia.* David R. Godine, Publisher, 1990.

[8] D. Parlett, *The Oxford History of Board Games.* Oxford University Press, 1999.

[9] G. Costikyan, "I have no words & I must design: Toward a critical vocabulary for games," in *Computer Games and Digital Cultures Conference Proceedings.* Tampere University Press, June 2002. [Online]. Available: http://www.digra.org/wp-content/uploads/digital-library/05164.51146.pdf

[10] J. Juul, "The game, the player, the world: looking for a heart of gameness," in *DiGRA - Proceedings of the 2003 DiGRA International Conference: Level Up*, 2003. [Online]. Available: http://www.digra.org/wp-content/uploads/digital-library/05163.50560.pdf

[11] K. Salen and E. Zimmerman, *Rules of Play: Game Design Fundamentals.* The MIT Press, 2003.

[12] T. Fullerton, C. Swain, and S. Hoffman, *Game Design Workshop: Designing, Prototyping and Playtesting Games.* CMP Books, 2004.

[13] J. Juul, *Half-real: Video games between real rules and fictional worlds.* The MIT Press, 2005.

[14] S. Björk, S. Lundgren, and J. Holopainen, "Game design patterns," in *DiGRA - Proceedings of the 2003 DiGRA International Conference: Level Up*, 2003. [Online]. Available: http://www.digra.org/wp-content/uploads/digital-library/05163.15303.pdf

[15] R. Hunicke, M. Leblanc, and R. Zubek, "MDA: A formal approach to game design and game research," in *In Proceedings of the Challenges in Games AI Workshop, Nineteenth National Conference of Artificial Intelligence.* AAAI Press, 2004, pp. 1–5. [Online]. Available: http://www.aaai.org/Papers/Workshops/2004/WS-04-04/WS04-04-001.pdf

[16] "Lemmings," Amiga, DMA Design, 1991.

[17] "Lemmings gameplay." [Online]. Available: https://youtu.be/IN-Ob5phQ1M

[18] "Limbo," Xbox Live Arcade, Playdead, 2010.

[19] "Limbo screenshots." [Online]. Available: https://www.trueachievements.com/game/LIMBO/screenshots

[20] "Inside," Xbox One, Playdead, 2016.

[21] "The 35 most beautiful roger deakins shots." [Online]. Available: https://filmschoolrejects.com/roger-deakins-most-beautiful-shots/

[22] D. Duprey, "Poster picks: Film versus marketing with blade runner." [Online]. Available: https://www.thatmomentin.com/poster-picks-film-versus-marketing-blade-runner/

[23] Plato, *The Republic*, 375 BC.

[24] I. Kant, *Critique of Pure Reason*, 1781.

[25] K. Craik, *The Nature of Explanation.* Cambridge University Press, 1943.

[26] R. M. Young, "The machine inside the machine: Users' models of pocket calculators," *International Journal of Man-Machine Studies"*, vol. 15, pp. 51–85, 1981.

[27] ——, "Surrogates and mappings: Two kinds of conceptual models for interactive devices," in *Mental Models*, D. Gentner and A. L. Stevens, Eds. Lawrence Erlbaum Associates, Inc., 1983, pp. 35–52.

[28] N. Staggers and A. F. Norcio, "Mental models: concepts for human-computer interaction research," *International Journal of Man-Machine Studies*, vol. 38, pp. 587–605, 1993.

[29] C. Argyris, "Teaching smart people how to learn," *Harvard Business Review*, vol. 4, no. 2, 1991.

[30] D. L. Meadows, W. W. Behrens, D. H. Meadows, R. F. Naill, J. Randers, and E. Zahn, *Dynamics of growth in a finite world.* Wright-Allen Press, 1974.

[31] B. Meyer, *Agile!* Springer, 2014.

[32] "The scrum framework poster." [Online]. Available: https://www.scrum.org/resources/scrum-framework-poster

[33] ZenHub, "Zenhub homepage." [Online]. Available: https://www.zenhub.com/

[34] GitHub, Inc., "Github homepage." [Online]. Available: https://github.com/

[35] T. G. Flouris and D. Lock, *Managing Aviation Projects from Concept to Completion.* Ashgate Publishing, 2009.

[36] "Principles behind the agile manifesto." [Online]. Available: https://agilemanifesto.org/principles.html

[37] "Manifesto for agile software development." [Online]. Available: https://agilemanifesto.org/

[38] "Git." [Online]. Available: https://git-scm.com/

[39] V. Driessen, "A successful git branching model." [Online]. Available: https://nvie.com/posts/a-successful-git-branching-model/

[40] Axosoft, "Gitkraken homepage." [Online]. Available: https://www.gitkraken.com/

[41] G. M. Toolkit, "Nintendo - putting play first | game maker's toolkit." [Online]. Available: https://youtu.be/2u6HTG8LuXQ

[42] Nintendo, "Nintendo homepage." [Online]. Available: https://www.nintendo.com/

[43] A. Cooper, R. Reimann, D. Cronin, and C. Noessel, *About Face: The Essentials of Interaction Design.* Wiley, 2014.

[44] H. L. Gantt, "Work, wages and profits," *The Engineering Magazine*, 1910.

[45] Adobe Inc., "Adobe XD homepage." [Online]. Available: https://www.adobe.com/products/xd.html#

[46] Microsoft Corporation, "Microsoft project homepage." [Online]. Available: https://www.microsoft.com/en-us/microsoft-365/project/project-management-software

[47] Adobe Inc., "Mixamo homepage." [Online]. Available: https://www.mixamo.com/#/

[48] Overleaf, "Overleaf homepage." [Online]. Available: https://www.overleaf.com

[49] Unity Technologies, "Unity homepage." [Online]. Available: https://unity.com/

[50] Epic Games, Inc., "Unreal engine homepage." [Online]. Available: https://www.unrealengine.com/en-US/

[51] J. Linietsky, A. Manzur, and contributors, "Godot homepage." [Online]. Available: https://godotengine.org/

[52] Raphael Ernaelsten, “Aura - volumetric lighting.” [Online]. Available: https://assetstore.unity.com/packages/tools/particles-effects/aura-volumetric-lighting-111664

[53] Archanor VFX, “Polygon arsenal.” [Online]. Available: https://assetstore.unity.com/packages/vfx/particles/polygon-arsenal-109286

[54] Synty Studios, “Polygon - sci-fi city pack.” [Online]. Available: https://assetstore.unity.com/packages/3d/environments/sci-fi/polygon-sci-fi-city-pack-115950

[55] ——, “Polygon - sci-fi space pack.” [Online]. Available: https://assetstore.unity.com/packages/3d/environments/sci-fi/polygon-sci-fi-space-pack-138857

[56] IMAscore, “Retro future music pack.” [Online]. Available: https://assetstore.unity.com/packages/audio/music/electronic/retro-future-music-pack-150623

[57] Cafofo, “Sci-fi sound pack.” [Online]. Available: https://assetstore.unity.com/packages/audio/sound-fx/sci-fi-sound-pack-154257

[58] Nicrom, “Water 2d tool.” [Online]. Available: https://assetstore.unity.com/packages/vfx/shaders/substances/water-2d-tool-35521

[59] Typodermic Fonts Inc., “Good times.” [Online]. Available: http://typodermicfonts.com/good-times/

[60] FlatIcons.net, “Flaticons.” [Online]. Available: https://flaticons.net/

[61] M. Sicart, “Defining game mechanics,” *Game Studies*, vol. 8, no. 2, 2008. [Online]. Available: http://gamestudies.org/0802/articles/sicart

[62] “The Legend of Zelda: Breath of the Wild,” Nintendo Switch, Wii U, Nintendo, 2017.

[63] “Angry Birds,” iOS, Rovio Entertainment, 2009.

[64] Unity Technologies, “Navigation and pathfinding.” [Online]. Available: https://docs.unity3d.com/Manual/Navigation.html

[65] A. Navarro Pérez and S. Soutullo Sobral, “Under surveillance - storage room level design time-lapse.” [Online]. Available: https://youtu.be/PbOtRs00tYc

[66] R. Bartle, “Hearts, Clubs, Diamonds, Spades: Players Who Suit MUDs.” [Online]. Available: http://mud.co.uk/richard/hcds.htm

[67] Google LLC, “Material Design.” [Online]. Available: https://material.io/design/

[68] Birdman, “Angry birds (pc gameplay - 1080p).” [Online]. Available: https://youtu.be/2BqfjGDsHUs

[69] A. Navarro Pérez and S. Soutullo Sobral, “Under surveillance - walkthrough.” [Online]. Available: https://youtu.be/ix5ictxuMVQ

[70] Itch.io, “Itch.io homepage.” [Online]. Available: https://itch.io/

[71] A. Navarro Pérez and S. Soutullo Sobral, “Under Surveillance.” [Online]. Available: https://adriannp57.itch.io/under-surveillance

[72] IEEE, “IEEE Conference on Games (CoG) 2020 homepage.” [Online]. Available: http://ieee-cog.org/2020/

[73] “Goal 5: Gender equality.” [Online]. Available: https://www.globalgoals.org/5-gender-equality

[74] “The global goals.” [Online]. Available: https://www.globalgoals.org/

[75] “Goal 4: Quality education.” [Online]. Available: https://www.globalgoals.org/4-quality-education

[76] “Goal 10: Reduced inequalities.” [Online]. Available: https://www.globalgoals.org/10-reduced-inequalities

A

The PEP Framework: A Formal Method to Design Consistent and Intuitive Physics-Based Games

The PEP Framework: A Formal Method to Design Consistent and Intuitive Physics-Based Games

Adrián Navarro Pérez
Department of Computer Science and Engineering
University of Gothenburg
Göteborg, Sweden
gusnavad@student.gu.se

Samuel Soutullo Sobral
Department of Computer Science and Engineering
University of Gothenburg
Göterborg, Sweden
gussoutsa@student.gu.se

***Abstract*—This paper presents the PEP framework, a formal method that guides the design and analysis of physics-based games. The developed work endeavors to facilitate these tasks by providing a set of defined steps, along with a common vocabulary all game designers can refer to. The developed formal process strives to ensure consistency in physics-based games, while simultaneously making more intuitive to the player the behaviors that emerge as a consequence of these games' laws of physics.**

***Index Terms*—Game Design, Game Research, Physics-Based Games, Philosophy, Psychology**

I. Introduction

A game is an artifact which supports a voluntary interaction carried out, within a formal independent transmedial system, between one or more users and the system itself, performing a finite number of different types of actions without expecting a productive outcome [1]–[13].

Games strive to produce meaningful gameplay when played, which can be achieved in vastly different ways. Physics-based games endeavor to do so by subjecting players' interactions or their consequences to a specific set of consistent laws of physics. This implies most, if not all, events emerging during gameplay are governed by these laws. As a consequence, these events must keep a solid level of consistency in their behaviors.

Maintaining consistency in these behaviors poses even more challenges to an already arduous task: game design. This demanding labor is further hindered by the fact that players are typically required to understand how the laws of physics make an impact in the game to be able to progress through it. This challenge is aggravated because it is not convenient nor practical to directly inform the player about these laws and each of their consequences.

To mitigate these challenges, inherent to physics-based games, it is necessary to have a common vocabulary and a formal method when tackling them. Formalization does not only provide a standard procedure to coordinate all the designers working in a project, but it also ensures games are designed according to an established process in which every design decision has a clear, justifiable reason. Consequently, the likelihood of solid, consistent game design and the probabilities of creating a successful product, are increased.

In response to this need, the PEP framework has been developed. The PEP framework, which stands for *physics*, *environment* and *player*, provides a common vocabulary and a formal method, both for designing and analyzing physics-based games, facilitating the completion of these tasks.

Along this paper, it is first provided a brief and general overview of the PEP framework architecture. Then, how the framework is grounded on previously existing concepts from the fields of philosophy and psychology is explained. Next, a more detailed overview of the PEP framework is given together with a specific set of instructions on how to apply it to physics-based games, regardless of their current development stage. Subsequently, real applications of the method are presented and examined, demonstrating its utility. To conclude, shortcomings and other future considerations regarding the presented work are stated.

Nevertheless, the development of this new formal method does not intend to set a new standard in the game industry. It aspires to be seen as a new support tool which can be used by game designers to ease their task in addition to already existing design methods [14], [15].

II. The PEP Framework

The PEP framework is a formal method that guides the design and analysis of physics-based games. It ensures consistency while making intuitive to the player the behaviors that emerge as a consequence of these games' physics.

According to the PEP framework, physics-based games can be broken down into three main components and three secondary components born from the interactions between the former. Fig. 1 illustrates how all these components relate to each other.

The three main components are:

- **Physics**: Set of rules governing the *environment*.
- **Environment**: Materialization of the current game state [10].
- **Player**: External user who interacts with the *environment*.

The three secondary components are:

- **Mechanics**: Methods invoked by the *player*, designed for interaction with the *environment* [16].
- **Phenomena**: Observable events or occurrences manifested in the *environment*, consequence of the *physics*.
- **Deductions**: *Player*'s knowledge gain regarding how the *physics* work derived from observed *phenomena*.

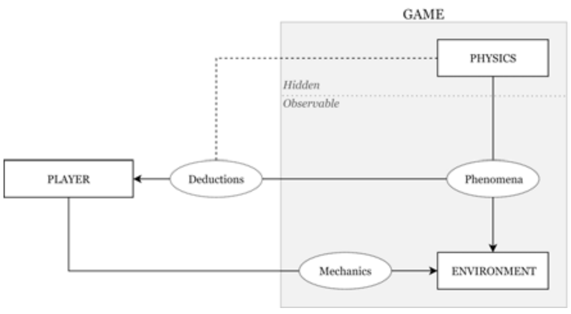


Fig. 1. The PEP framework architecture.

III. Background

Research in the fields of philosophy and psychology has been performed to develop a framework that can be applied to analyze and design physics-based games. Concerning philosophy, the study has been focused on epistemological currents, while in the field of psychology, efforts have been concentrated in the foundations of cognition.

A strong correlation between the real world, from an epistemological perspective, and the main principles of physics-based games was found. They share the same structure of consistent events that emerge according to the laws of physics in both scenarios. Moreover, humans learn in both cases how these laws work primarily by observing such events, e.g., the humankind learned about gravity because they were able to perceive its effects and impact on the surroundings.

Simultaneously, the envisioned framework intended to take into consideration not only the consistency of physics-based games but also the player's learning process during gameplay. Thus, the terms cognition and reasoning became relevant to the research process.

The *Allegory of the Cave* by Plato, part of the Book VII from his Socratic dialogue, The Republic [17], and the *Critique of the Pure Reason* by Kant [18], came out as profoundly influential philosophical theses to determine and specify the different components of the PEP framework. At the same time, two tightly related psychological concepts, namely, *mental model* and *double-loop learning*, became essential when developing the cognitional aspects of the framework.

A. Allegory of the Cave

The *Allegory of the Cave* is one of the most important and widely-known allegories in the annals of philosophy. Part of its acclaim comes from successfully explaining the pillars of Platonism in a simple, yet profound way. It aims to explain the situation of the human perception with respect to true knowledge and how to acquire it over philosophical reasoning.

In this parable, Plato describes a cave inhabited by a group of people, prisoners since they were born, who can only see the walls of the cave. Right behind them, there is another wall and a corridor they cannot see, which is populated by another group of people who walk around carrying a set of objects. Behind the prisoners, there is also a bonfire which enlightens the corridor, illuminating the wall of the cave the prisoners can see. Therefore, the prisoners observe on the walls the shadows projected by the set of objects which the other group of people is carrying around.

Prisoners consider as a legitimate truth the projected shadows they see on the wall. For them, that is their genuine reality since they are not aware of what is happening out of their own reach and understanding. They do think the real world is limited to those shadows, which are just a small consequence of a greater whole: a world where light interacts with opaque objects generating shadows.

The conceptual representation of the *Allegory of the Cave* in Fig. 2 illustrates how the most important concepts of the parable relate to one another. This diagram aims to synthesize the allegory to only what is strictly relevant for the PEP framework development. These concepts are related to the framework in Sec. IV.

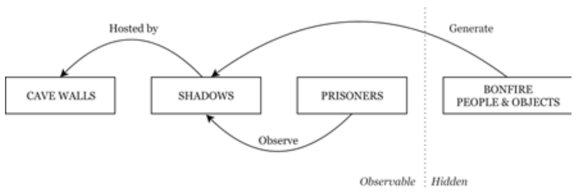


Fig. 2. Conceptual representation of the *Allegory of the Cave*.

B. Critique of Pure Reason

Critique of Pure Reason, originally published in 1781, is one of the most relevant books written by the German philosopher Immanuel Kant. In this work, Kant explores and determines the boundaries of metaphysics, a fundamental branch of philosophy that deals with the study of beings and their properties, principles, causes and fundamentals of their existence.

Before Kant published his work, there were two main epistemological currents: rationalism and empiricism. Kant concluded that both currents were problematic. Pure rationalism ends up in dogmatism, i.e., use of the reason without examining its limits. On the other hand, pure empiricism denies all the knowledge which cannot be tested empirically, leading to radical skepticism. To solve these limitations, Kant developed his own thesis by examining pure reason, its limits and incorporating empirical aspects to his work.

Some of the most significant concepts Kant developed in *Critique of Pure Reason* are closely related to multiple components of the PEP framework. These concepts and the relations between them are conceptually represented in Fig. 3. Their influence on the PEP framework is explained in Sec. IV.

Kant refers to transcendental as everything related to knowledge previous to the experience, such as time, space and fundamentals of logic. This knowledge, is often referenced as *a priori* knowledge by the philosopher. *A priori* knowledge is

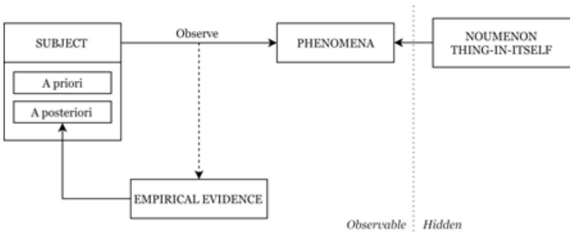


Fig. 3. Conceptual representation of the relevant concepts from the *Critique of Pure Reason*.

inherent to the subject and, by itself, cannot be used to acquire new knowledge.

On the other hand, *a posteriori* knowledge is derived from the experience, often providing novel knowledge to the subject. Furthermore, the only way to acquire and verify such knowledge is through *empirical evidence*, obtained by observation. However, *a posteriori* knowledge is not universal. It may be false in any case other than the observed.

Kant also argues that reality in itself is unknown to subjects. The *thing-in-itself*, or *noumenon*, cannot be perceived by subjects as it is inaccessible to them. Consequently, they can only perceive reality through *phenomena*, which is how it subjectively appears to them. According to Kant, questions such as wondering "the how" are irrelevant. A better conception of reality would be asking how it is perceived by a specific subject. Understanding reality implies shaping it with the subject's *a priori* knowledge.

C. Mental model and double-loop learning

A *mental model* is a user's internal and structured representation of a system. It is originated or modified from the interaction between the user and external events which help to guide the user's actions and to interpret the system's behavior. [19]–[22].

When users address a specific goal, the *mental model* might need to be modified to gain a reactive and deep understanding of their surroundings to accomplish such goal. This modification relies upon the user's individual experience (reflexive thinking). This cognitive process is known as *double-loop learning* [23].

The most important part of *double-loop learning* for the development of the PEP framework is the *mental model*. The graph shown in Fig. 4 illustrates *double-loop learning* by representing an individual observing its surroundings and gathering information. This information is then used to update their *mental model*, i.e., their interpretation of the real world. The rest of the elements show all the information being processed by the individual, ultimately used to make decisions and accomplish goals that may change and have an impact on the world.

IV. The Framework in Detail

This section describes each of the PEP framework components in detail while establishing explicit relations between

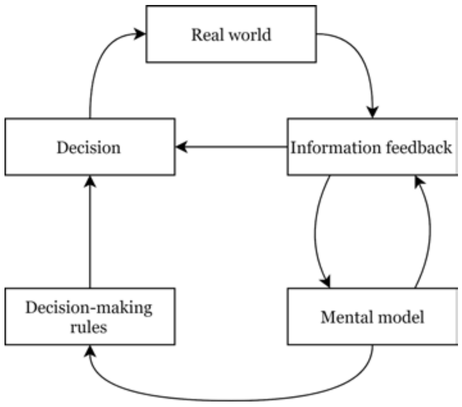


Fig. 4. Conceptual representation of double-loop learning adapted from [24].

these concepts and the ones previously explained in Sec. III. Fig. 5 provides a summary of these relations, further explained in the following subsections.

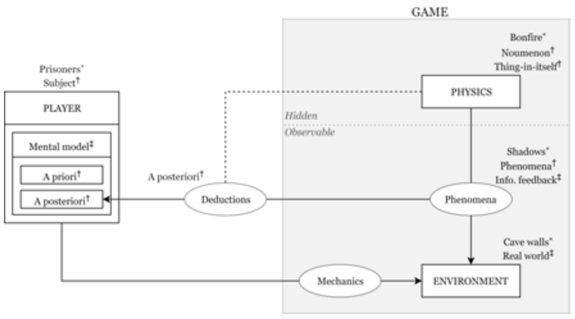


Fig. 5. The PEP framework architecture in detail. Related Plato's (*), Kant's (†) and mental model and double-loop learning (‡) concepts are shown.

A. Physics

Physics are the set of rules governing the *environment* of any physics-based game. This includes, but it is not limited to, Newton's laws of motion, thermodynamics' laws and the electromagnetic force.

However, there are video games featuring *physics* which are not present in the real universe, nor do they work as one might expect, leading to behaviors only present in these games. This is the case of *The Legend of Zelda: Breath of the Wild* [25] when Link uses the Stasis Rune to temporarily freeze an object in time, storing potential energy while the effect lasts. Once it is over, all the corresponding kinetic energy is then liberated.

In relation to the *Allegory of the Cave*, the bonfire is equivalent to the *physics*. Since the prisoners cannot see the bonfire, the interpretations they can make about the shadows' origin are potentially wrong. Similarly, the *physics* are hidden to the *player*, who can only interpret how they work by observing the *environment* and the emerging *phenomena*. *Physics* are the reason why *phenomena* emerge in the *environment* as the bonfire is the reason why shadows are projected on the wall.

Regarding Kant's philosophy, the noumenon, or thing-in-itself, is equivalent to the *physics*. The noumenon cannot be perceived by the subject, or in the case of games, the *physics* cannot be directly observed by the *player*. But notwithstanding this, the *physics* govern the *environment*. They are the noumenon which instantiates the *phenomena*.

B. Environment

The *environment* is the materialization of the current game state. It is the whole game world including every single game object, characters and, when it is present, the *player*'s avatar, which is the *player*'s representation in the *environment*. Given *Angry Birds* [26], later renamed as *Angry Birds Classic*, as an example of physics-based video game, the *environment* includes all birds, pigs, the slingshot, wooden planks, glass blocks, etc.

In the *Allegory of the Cave*, the walls of the cave correspond to the *environment*. It is in these observable walls that the shadows remain visible to the prisoners. Analogously, in the PEP framework, the *environment*, observable by the *player*, hosts emerging *phenomena* during gameplay.

Additionally, double-loop learning's real world is equivalent to the *environment* as they are both observable elements in which feedback and *phenomena* emerge, respectively.

C. Player

The *player* is an external user to the game who interacts with it via the use of game *mechanics*.

Regarding the *Allegory of the Cave*, the prisoners and the *player* are correlative. The prisoners can only perceive the shadows projected on the walls while the bonfire and the people carrying the objects are not visible to them. Correspondingly, the *player* can only perceive the emergent *phenomena* in the *environment* while the *physics* remain hidden to them.

In connection with Kant's philosophy, the subject is the *player*, who can perceive *phenomena*, gathering empirical evidence. From this evidence, a posteriori knowledge, in form of *deductions* on how the *physics* work, is built.

All the *player*'s a posteriori knowledge modifies and improves their mental model of the *physics* and how they work. This mental model is also supported by the *player*'s a priori knowledge. From a Kantian perspective, a priori knowledge is not derived from any experience in the real world. Similarly, from the PEP framework perspective, the *player*'s a priori knowledge is not derived from any experience within the game, but it can be derived from any external experiences to the game itself, such as the ones lived in the real world or while playing other games. How gravity works in the real world is an example of a *player*'s a priori knowledge. The *player* makes good use of this knowledge to shape an early mental model of how gravity might work in the game. Eventually, the gameplay makes the *player* to compare how gravity works both in the real world and in the game, leading to a posteriori knowledge and a more accurate mental model of how it behaves in the latter.

D. Mechanics

The *mechanics* are methods invoked by the *player* to interact with the game. Nevertheless, this interaction can be carried out only with the *environment*. Performing *mechanics* can potentially change the *environment* and, equivalently, the game state.

E. Phenomena

The *phenomena* are observable events or occurrences which manifest in the *environment* as a direct consequence of the game's *physics*. These visible manifestations are perceived by the *player*.

Examples of *phenomena* in physics-based games are falling objects due to gravity, burning objects due to fire and thermodynamics' laws and the propagation of electricity in conductive materials due to the fundamental laws of electricity. The different types of *phenomena* present in these types of games and the *physics* ruling them are only constrained by designers' creativity.

Phenomena are the only observable part of a game related to the *physics*. Thus, it is only by thoroughly observing these *phenomena* and making their own *deductions* that the *player* can understand to some extent how the *physics* work in the game.

In the *Allegory of the Cave*, the projected shadows on the wall are equivalent to the *phenomena*. These shadows are a visible manifestation consequence of the bonfire in conjunction with the group of people carrying objects. The prisoners perceive the shadows, which are the only event they can interpret to understand the rules of the world they inhabit. Likewise, the *player* can only observe *phenomena* which manifest in the *environment* to better understand the *physics* governing it.

In relation to Kant's philosophy, how Kant describes *phenomena* is analogous to how it is defined in the PEP framework. Kant's *phenomena* emerge from the noumenon or thing-in-itself constituting observable events in the same way PEP framework's *phenomena* emerge as a consequence of the *physics*.

According to what the information feedback means in double-loop learning, a strong connection between such information and the *phenomena* is present. Information feedback is obtained from facing goals and challenges within the real world which lets people learn from their own ways to approach them and their results, continuously updating their mental models. The same goes for the *phenomena* which help the *player* to learn about their surrounding *environment*, honing their mental model of the *physics* of the game.

F. Deductions

Deductions are the *player*'s knowledge gain regarding how the *physics* work according to the observed *phenomena*. All this knowledge helps the *player* to form a mental model on how the *physics* might work and which *phenomena* might emerge from applying certain *mechanics* to the *environment*. The *player* will never know for sure how close their mental model got to the actual game's *physics*. However, a very close approximation is to be expected if the game is consistent and intuitive to the *player* at all times.

As for Kant's philosophy, a posteriori knowledge is considered as a set of *deductions* by the PEP framework. The *player*'s mental model is updated according to the *phenomena* they observe and the empirical evidence they gather. Alike a posteriori knowledge, *deductions* are not universal, meaning they may perfectly explain the observed *phenomena* but it is not enough to fully understand the underlying nature of the *physics*.

To conclude, the acquisition of knowledge is an iterative process in which the *player* gets a better understanding of the *physics* the more *phenomena* they observe, and the more *deductions* they make from them.

V. The Framework Methodology

This section provides a list of steps to follow when applying the PEP framework from two different perspectives: design and analysis.

From the design point of view, the procedure starts by creating the *phenomena* and then deriving the *environment* from them. These first two steps ensure consistency between both of them, since every *phenomenon* is thereafter supported by at least one element from the *environment*. Next, the *physics* and game *mechanics* are created by taking into account the existing *phenomena* and *environment*. This guarantees a set of *physics* covering all the possibilities of the *phenomena* and *environment*, while at the same time every *mechanic* supports the emergence of *phenomena*. As the final step, the *deductions* a *player* might make during gameplay are designed to control how *players* learn about the game's *physics*.

From the analysis perspective, the method explains how to make use of the framework as a formal approach to analyze physics-based games to review their consistency. It also helps game designers to detect problems and avoid repeating the same mistakes. The analysis starts by identifying in no particular order the game *mechanics*, the *phenomena* and the *environment* elements. These elements are then validated to check whether they are consistent with respect to each other. Finally, the *deductions* a *player* can make are evaluated to check if the *player* might be overwhelmed when learning how the game's *physics* work during gameplay.

A. Design

This subsection explains the formal process on how to use the PEP framework as part of the design and development of new physics-based games. This process consists of the following steps:

1) Design phenomena: In this creative task, game designers propose *phenomena* which will be present in their new physics-based game depending on what type of game they want to create.

2) Design environment: Taking as a starting point the *phenomena* designed in the previous step, an *environment* is derived from them. How designers want the *environment* to be like has a considerable impact on how important the *player*'s a priori knowledge will be in the game. If the *phenomena* are typically found in the real world or other games played by the *player*, then the *player* will make use of a substantial amount of their a priori knowledge to understand how the *physics* work. On the other hand, if the *player* has rarely or never witnessed the *phenomena* emerging in this particular *environment*, the *player* will not be able to apply any a priori knowledge. Thus, the *player* will only rely on their a posteriori knowledge gained during gameplay through *deductions*.

3) Formalize physics: Once the *phenomena* and the *environment* have been defined, the *phenomena* behaviors in such *environment* are generalized by defining laws of *physics*. These set of laws will rule all the *environment* and all its emerging *phenomena*.

4) Design mechanics: This step is interchangeable with the previous one, since having the *phenomena* and the *environment* defined suffices to start designing game *mechanics*. In order to design the *mechanics*, how the *player* will interact with the *environment* must be specified. These *mechanics* must support the *phenomena* by allowing the *player* to directly or indirectly originate, alter or bring to an end *phenomena*.

5) Design deductions: All the previous steps aim to improve consistency in physics-based games by ensuring the *phenomena*, the *environment* and the *mechanics* exist for a justifiable reason. However, besides consistency, the framework also strives to take into account how the *player* can learn and understand how the *physics* work in these games. In this regard, in order to ensure a game in which its *physics* are intuitive to the *player*, the amount of *deductions* the *player* needs to make must be limited during gameplay. This can be achieved in two different ways. First, by designing more familiar *phenomena* and *environment* to the *player*, so more a priori knowledge is used. Secondly, by designing game levels where all *phenomena* and *environment* are gradually introduced, to avoid overwhelming situations to the *player* in terms of learning.

B. Analysis

This subsection describes the formal process on how to make use of the PEP framework as part of the analysis to break an existing physics-based game down. It consists of the following steps:

1) Identify the phenomena, the environment and the mechanics: All the *phenomena*, the *environment* elements and the *mechanics* are listed. Since the game is already designed, these can be identified in any particular order.

2) Validate the phenomena and the environment: The *phenomena* identified in the previous step must be validated by checking whether or not the *environment* supports such *phenomena*. In any state of the game, given the same *environment* elements, it must be checked if the same *phenomena* emerge. If they do not, a lack of consistency is then detected.

3) Validate the mechanics: The *mechanics* identified in the first step must be validated by checking if they directly or indirectly support the *phenomena*. *Mechanics* which do not fulfill this condition, have no relevance to the gameplay, hence, they have no positive impact on the game's consistency.

4) Evaluate the deductions: The previous steps examine the consistency of a physics-based game by checking whether its proper *phenomena* emerge, supported by its *mechanics*, according to its *environment*. Nevertheless, how the *player*'s cognitive process on understanding how the *physics* work in these games is also tackled in this analysis process. Therefore, the *player*'s *deductions* must be evaluated too. To perform this task, it must be determined first the a priori knowledge most of the *players* have available before playing the game for the first time. Then, it must be studied at which points of the game the *phenomena* emerge. Every *phenomenon* which cannot be explained by the *player*'s a priori knowledge will lead them to make *deductions*, establishing a posteriori knowledge about how the game's *physics* work. The more gradually the *player* needs to make *deductions*, the less overwhelming the game becomes when learning about it and its *physics*.

VI. The Framework Applied

In this section, the methodology described and explained in Sec. V is applied to two different physics-based games. First, the design process is applied to *Under Surveillance*, a physics-based computer video game currently under development by the authors. Secondly, the analysis process is applied to *Angry Birds*, a physics-based mobile video game released in 2009.

A. Design: Under Surveillance

Under Surveillance is a 2.5D physics-based puzzle adventure video game. The *player* takes control of a mysterious robot who is able to use and manipulate electricity, fire and water in order to solve a series of puzzles while progressing in a dystopian world.

To design *Under Surveillance*, the PEP framework has been applied, following the formal process described in Sec. V-A:

1) Design phenomena: Right from the beginning, the authors envisioned a game featuring electricity, fire and water behaving as they do in the real world. Thus, the following *phenomena* were proposed:

- **Electricity**: Propagation through conductive materials and interaction with electrical devices.
- **Fire**: Propagation through burnable materials, smoke generation and extinction on contact with water.
- **Water**: Movement according to fluid dynamics, buoyancy of different objects, vaporization due to hot temperatures and freezing due to cold temperatures.

2) Design environment: The authors envisioned a game in which the challenge was not to figure out how electricity, fire and water behave, but rather how to use these elements to solve the puzzles present along the way. Therefore, letting the *player* make use of their a priori knowledge as much as possible was a must. Consequently, elements typically found in the real world interacting with the aforementioned *phenomena* were designed as part of the *environment*. These *environment* elements, classified by whether they typically interact with electricity, fire or water are:

- **Electricity**: Metallic and non-metallic objects which conduct and isolate electricity, respectively, and electrical devices that can be turned on or off.
- **Fire**: Objects made of flammable materials, such as wood, and nonflammable materials, such as steel.
- **Water**: Shapes allowing the fluid dynamics *phenomena* to emerge, such as slopes or empty pools, and objects with different buoyancies which might float or sink when put in water.

3) Formalize physics: According to the aforementioned designed *phenomena* and *environment*, the *physics* were generalized. Since the electricity, fire and water emulate their own behaviors in the real world, simplifications of actual laws of *physics* were considered.

4) Design mechanics: Given the need to support the *phenomena* with simple, yet powerful *mechanics*, the *player* is provided with the possibility to interact with the *environment* by throwing electricity, fire or water to any *environment* element.

5) Design deductions: So far, the *phenomena* and the *environment* were designed so as much as possible a priori knowledge is used by the *player*. Moreover, the *player* is not able to use electricity, fire and water from the beginning of the game, but these are gradually introduced as they progress. These two design decisions sort and limit the amount of *deductions* the *player* needs to make in order to gain a posteriori knowledge. Thus, the *player* can now intuitively learn and understand how the formalized *physics* work as the *deductions* they might make have been adjusted, avoiding overwhelming situations in terms of learning.

B. Analysis: Angry Birds

Angry Birds is a 2D puzzle physics-based mobile video game created by *Rovio Entertainment* and originally published in 2009. In *Angry Birds*, the *player* makes use of a slingshot to shoot different birds to multiple fortresses to make them collapse while neutralizing the pigs located at them.

In order to analyze *Angry Birds*, the PEP framework analysis process has been applied by following all the steps explained in Sec. V-B:

1) Identify the phenomena, the environment and the mechanics: A list with all the *phenomena*, the *environment* elements and the *mechanics* has been tailored:

- **Mechanics**: Shoot birds, use the birds' special abilities.
- **Environment**: Earth-like world, planks made of glass, wood and stone, TNT boxes, pigs and birds.
- **Phenomena**: Parabolic trajectories, collisions, damage, destruction and explosions.

2) Validate the phenomena and the environment: All the aforementioned identified *phenomena* are then validated by checking whether they are supported by the *environment* elements or not:

- **Parabolic trajectories**: All the objects present in the *environment* behave consistently. They always describe trajectories governed by the same laws of *physics*.
- **Collisions**: All the objects present in the *environment* collide with each other in a consistent manner according to the game's formalized *physics*.
- **Damage and destruction**: Planks consistently take damage when they receive an impact by other objects of the *environment* and they are destroyed once they have received enough damage. Under the same conditions and given the same impact, a plank always receives the same amount of damage. In order to provide a bigger variety of this kind of *phenomena*, planks are made of different materials, differing in the maximum amount of damage they can take before they are destroyed.
- **Explosions**: TNT boxes explode when they receive an impact by a different object with a force greater than a specific and constant threshold. The damage of these explosions to the surrounding objects is always the same, according to the distances to them.

3) Validate the mechanics: All the previously identified *mechanics* are validated by checking whether they directly or indirectly support the *phenomena*:

- **Shoot bird**: Shooting birds is the main game *mechanic*. By shooting a bird, the *player* can make all the aforementioned identified *phenomena* emerge in a direct way: parabolic trajectories, collisions, damage and destruction when birds hit fortresses and explosions when birds hit a TNT box. Also, these *phenomena* can emerge indirectly due to the chain reaction produced when fortresses break and fall over other *environment* elements. Thus, this *mechanic* supports the *phenomena*.
- **Use bird's special ability**: All the birds' special abilities are designed to increase the potential amount of damage dealt to fortresses. Therefore, this *mechanic* supports the *phenomena*.

4) Evaluate the deductions: Angry Birds is designed to be an intuitive game. This is achieved both by making use of as much a priori knowledge as possible and by designing levels so new *phenomena*, *environment* elements, and game *mechanics* are steadily introduced.

The following design decisions to support the use of a priori knowledge have been found out when performing this analysis:

- **Earth-like world**: Many *phenomena* simulated in the game behave as they do in the real world. Daily, parabolic trajectories are witnessed in the real world due to the action of gravity. By creating an *environment* reassembling the Earth, the *player* expects to take into account the gravity force before shooting a bird, even if they have never played *Angry Birds* before.
- **Birds**: Since *Angry Birds* is entirely built around the idea of shooting projectiles following parabolic trajectories, these projectiles spend most of their time in mid-air. Naturally, these projectiles became an element present on Earth's nature which is usually found flying around: birds.
- **Slingshot**: Having birds as the game's projectiles to describe parabolic trajectories is not intuitive enough to the *player* as birds do not fly like that. However, when a slingshot is added as part of the *environment* when shooting a bird, the *player* can then safely assume all birds shot with such slingshot will follow a parabolic trajectory, even if they have never interacted with the game's slingshot before.
- **Planks' materials**: Glass, wood and stone are familiar materials and so are their general properties. Furthermore, they are easy to distinguish from each other. This is part of *Angry Birds*' *environment* design, with glass planks that are easily destroyed, stone planks which are hard to break, and wood planks which resilience is somewhere in between the other two.
- **TNT boxes as explosion triggers**: The relationship between TNT and explosions is very common in games as well as it is in the real world. Thus, when the *player* identifies a TNT box in the *environment*, they expect to be able to destroy it and cause a big explosion *phenomenon*. *Angry Birds* is not an exception to this.

On the other hand, there are also aspects of *Angry Birds* which the *player* needs to learn by playing, as their a priori knowledge is not enough. Nonetheless, these aspects are carefully and gradually introduced to the *player*. Thus, the amount of *deductions* made during gameplay is limited, not overwhelming the *player* with a burst of new concepts early in the game. Thanks to this design approach, a posteriori knowledge can more easily be built by the *player* around these concepts.

One of the most relevant and illustrative examples of how these concepts are introduced is how new birds are presented to the *player*. First, when a new type of bird is introduced, it is always done once the *player* has completed a fair amount of levels in which no new bird has appeared. Secondly, in the levels where a new type of bird appears, it is always done by following a specific structure. This structure consists in letting the *player* shoot several instances of the same new type of bird and none of the others. This improves the quality of the *deductions* the *player* makes when experimenting with a new element from the *environment*, such as the new bird, by letting the *player* trigger more instances of the same *phenomenon*, leading to a more accurate a posteriori knowledge.

VII. Limitations and Challenges

Some limitations must be taken into account when making use of the PEP framework.

First, as mentioned in Sec. I, this work does not intend to become the only method to apply when designing or analyzing physics-based games. The framework does not aim to help to face all the challenges that need to be tackled when creating a game or analyzing it. For example, the PEP framework does not take into consideration the *player*'s desirable emotional responses (aesthetics) during gameplay, which is addressed by other formal approaches such as [15].

Secondly, extensive testing of the PEP framework has not been conducted. Consequently, how well the framework performs at edge cases is yet to be demonstrated.

Lastly, as for the generation of *deductions* during gameplay, it has been assumed the *player* has no previous experience with the game. This might not be the case in some scenarios in which the *player* has watched other people playing the same game or they have gathered information about the game in any other way.

VIII. Conclusion and Further Work

This paper has introduced, explained and applied the PEP framework, a formal method to design consistent and intuitive physics-based games. An overview, as well as a detailed description of its architecture, have been presented along with formal procedures that describe how to apply the framework when designing and analyzing physics-based games. Finally, the limitations and challenges of the developed work have been pointed out.

While developing the framework, it was postulated that without changing its core architecture, but only slightly modifying some of its components, it could also be used to design and analyze games that are not necessary physics-based.

Although this aspect of the PEP framework has not thoroughly been explored, the authors realized that most of its components are common to most games. *Environment*, *player* and *mechanics* are present in every game by definition. As for the *physics*, these could be generalized to all the rules governing a game. Hence, the concept of *phenomena* would still be present in any game, not as the emergence of observable occurrences as a consequence of the *physics*, but rather as a consequence of the game rules. Conclusively, the *deductions* would be related to how these rules work.

Nonetheless, games in which the aforementioned aspect of the PEP framework does not apply do exist. When all the game rules are known by the *player* beforehand, which is common in board games, *deductions* are not present since *players* are not necessarily expected to learn the rules during gameplay.

Acknowledgment

This work was developed by Adrián Navarro Pérez and Samuel Soutullo Sobral with the supervision of Marco Fratarcangeli. Adrián acknowledges the financial support from *Fundación Margit y Folke Pehrzon*, *Svensk-Spanska Stiftelsen / The Swedish-Spanish Foundation* and *Adlerbertska Foreign Student Hospitality Foundation* and Samuel acknowledges the financial support from *Fundación Barrié*.

References

[1] O. Morgenstern and J. von Neumann, *Theory of Games and Economic Behavior*. Princeton University Press, 1944.
[2] J. Huizinga, *Homo Ludens: A Study of the Play-Element in Culture*. Routledge & Kegan Paul Ltd, 1949.
[3] R. Caillois, *Man, Play, and Games*. Thames & Hudson, 1961.
[4] C. C. Abt, *Serious Games*. Viking Press, 1970.
[5] E. M. Avedon and B. Sutton-Smith, *The Study of Games*. John Wiley & Sons, 1971.
[6] C. Crawford, *The Art of Computer Game Design: Reflections of a Master Game Designer*. Osborne/McGraw-Hill, 1984.
[7] B. Suits, *The Grasshopper: Games, Life and Utopia*. David R. Godine, Publisher, 1990.
[8] D. Parlett, *The Oxford History of Board Games*. Oxford University Press, 1999.
[9] G. Costikyan, “I have no words & I must design: Toward a critical vocabulary for games,” in *Computer Games and Digital Cultures Conference Proceedings*. Tampere University Press, June 2002. [Online]. Available: http://www.digra.org/wp-content/uploads/digital-library/05164.51146.pdf
[10] J. Juul, “The game, the player, the world: looking for a heart of gameness,” in *DiGRA - Proceedings of the 2003 DiGRA International Conference: Level Up*, 2003. [Online]. Available: http://www.digra.org/wp-content/uploads/digital-library/05163.50560.pdf
[11] K. Salen and E. Zimmerman, *Rules of Play: Game Design Fundamentals*. The MIT Press, 2003.
[12] T. Fullerton, C. Swain, and S. Hoffman, *Game Design Workshop: Designing, Prototyping and Playtesting Games*. CMP Books, 2004.
[13] J. Juul, *Half-real: Video games between real rules and fictional worlds*. The MIT Press, 2005.
[14] S. Björk, S. Lundgren, and J. Holopainen, “Game design patterns,” in *DiGRA - Proceedings of the 2003 DiGRA International Conference: Level Up*, 2003. [Online]. Available: http://www.digra.org/wp-content/uploads/digital-library/05163.15303.pdf
[15] R. Hunicke, M. Leblanc, and R. Zubek, “MDA: A formal approach to game design and game research,” in *In Proceedings of the Challenges in Games AI Workshop, Nineteenth National Conference of Artificial Intelligence*. AAAI Press, 2004, pp. 1–5. [Online]. Available: http://www.aaai.org/Papers/Workshops/2004/WS-04-04/WS04-04-001.pdf
[16] M. Sicart, “Defining game mechanics,” *Game Studies*, vol. 8, no. 2, 2008. [Online]. Available: http://gamestudies.org/0802/articles/sicart
[17] Plato, *The Republic*, 375 BC.
[18] I. Kant, *Critique of Pure Reason*, 1781.
[19] K. Craik, *The Nature of Explanation*. Cambridge University Press, 1943.
[20] R. M. Young, “The machine inside the machine: Users’ models of pocket calculators,” *International Journal of Man-Machine Studies”*, vol. 15, pp. 51–85, 1981.
[21] ——, “Surrogates and mappings: Two kinds of conceptual models for interactive devices,” in *Mental Models*, D. Gentner and A. L. Stevens, Eds. Lawrence Erlbaum Associates, Inc., 1983, pp. 35–52.
[22] N. Staggers and A. F. Norcio, “Mental models: concepts for human-computer interaction research,” *International Journal of Man-Machine Studies*, vol. 38, pp. 587–605, 1993.
[23] C. Argyris, “Teaching smart people how to learn,” *Harvard Business Review*, vol. 4, no. 2, 1991.
[24] D. L. Meadows, W. W. Behrens, D. H. Meadows, R. F. Naill, J. Randers, and E. Zahn, *Dynamics of growth in a finite world*. Wright-Allen Press, 1974.
[25] “The Legend of Zelda: Breath of the Wild,” Nintendo Switch, Wii U, Nintendo, 2017.
[26] “Angry Birds,” iOS, Rovio Entertainment, 2009.