Project - Interactive Systems

E1+E2+E3+E4+E5+E6 - Final Stage

Student: Racz Ervin
Email: rcz.ervin@gmail.com
Evaluators: Sergiu Redeca, Bogdan Ban

Table of Content

Introduction

The computer- or video-game industry has grown in the past few decades from focused markets to a big mainstream market. As a sign of this growth, we can notice that eSports are massively popular these days − by some metrics, more popular than conventional sports − and they are just now at the beginning. The game industry and eSports have changed the perspectives, by now, not only focusing on young audience, but on adults, capitalizing worldwide on a huge audience by cutting across geographic, class and cultural differences.

What is gaming?

Well, the word “game” can be used in many contexts. According to an online dictionary, the noun means the followings:

- covers an activity, an amusement or pastime
- the material or equipment used in playing certain games
- a competitive activity involving skill, chance, or endurance on the part of two or more persons who play according to a set of rules, usually for their own amusement or for that of spectators

And according to wikipedia, video game is “an electronic game that involves interaction with a user interface to generate visual feedback on a two- or three-dimensional video display device such as a TV screen, mobile devices, virtual reality headset or computer monitor”.

Fig 1. People playing a large scale version of the iconic Pong video game at the National Videogame Museum

So, gaming is the activity when you “do games”. But the word gaming is much more associated with video games then regular child games or board games, therfore when someone talks about gaming, everybody knows that some kind of video game is involved in the discussion.

Controllers

To create an interaction between the video game and the human, video games can use several type of input devices to translate human actions for a game. The most common game controllers are keyboard and mouse for “PC gamers”, but another popular choice is a gamepad. Gamepads can be made specifically for game consoles. Other game controllers are made for specific game types like racing wheels, light guns or dance pads. Digital cameras can also be used as controllers. As technology continues to advance, more can be added onto the controller to give the player a more immersive experience when playing different games.

Genres

A video game, like other forms of media, may be classified into groups, generes. Video game genres are used to categorized video games based on their gameplay interaction rather than visual, content or narrative differences, unlike other works of fiction such as films or books. For example, a shooter game is still a shooter game, regardless of its virtual world, which can be presented as an environment like our real world or a fantasy world from the future. The list of video game genres is not short, and you can gather more information about this topic on wikipedia, where people tries to maintain a list of these game categories.

Puzzle video games

Puzzle video games is the targeted category of my project. Puzzle video games make up a unique genre of video games that emphasize puzzle solving. Generally speaking, a puzzle is a game or problem that tests a person's ingenuity or knowledge. In the game, the solver is expected to put pieces together in a logical way, in order to get to the correct or fun solution of the puzzle.

The types of puzzles can test many problem-solving skills including logic, pattern recognition, sequence solving, and word completion. The player may have unlimited time or infinite attempts to solve a puzzle, or there may be a time limit, or simpler puzzles may be made difficult by having to complete them in real time, like in tetris.

1. Project Proposal

1.1 Description

Name: Maze of Numbers
Domain: 3D PC game
Genre: puzzle game
Controllers: keyboard + mouse
Game Camera: first-person perspective
Gameplay: In *Maze of Numbers*, the player controls the unnamed protagonist form a first-person perspective. The player finds himself in the middle of a maze, in a junction of three corridor, from where he can start to walk in three different directions. The main game mechanics of this puzzle game is the ability to maneuver the protagonist around the spaces from one junction to another, where a variety of level elements can be found making distinction between junctions (e.g. lights with different colors, tables with numbers drawn on it, rebuses, some kind of objects) each of them hiding a number. After moving around a while, the player should notice that this game is different from the others regarding the typical notions of euclidean space, because the maze will present itself as a never ending bended space. The maze should brake down all the expectations after noticing that following only our intuition can easily result arriving to a point where we just started. The goal of the player is to find the exit door. Passing through that door means completing the level.

1.2. Audience

Solutions of puzzles often require the recognition of patterns. People with a high level of inductive reasoning aptitude (measures how well a person can identify patterns) may be better at solving such puzzles than others. But puzzles based upon inquiry and discovery may be solved more easily by those with good deduction skills. Deductive reasoning improves with practice. The market of puzzle games is focused on people who like solving logistical or mathematical problems and since the game that I proposed is based on mathematical problems, the only people who will like this game are those that are familiar with high-school grade mathematics.

1.3. Sources of Inspiration

Portal

Portal is a first person game, which offers puzzle adventures with unique mechanics for puzzle fans to solve. The Portal series was born in 2007 and received plenty of praise for its unique puzzle solving mechanics and the way in which players advanced through the story. The original game was really short in length and left players wanting more which led to the release of a sequel in 2011 which introduced co-operative mechanics into the series.

Fig 2. In-game photo: Portal

Antichamber

Antichamber is a first person puzzle game that will mess with the way you perceive video games and in order to succeed you'll have to rid yourself of preconceived notions relating to games that you have played. The game is a psychological adventure that challenges what you know about the first person genre. Players will have to take a step back, carefully examine the environment and solve the cryptic clues that the signs provide. As you do this for more puzzles the typical expectations of what you expect to happen will break down and be reshaped into something new.

In Antichamber the player will explore a number of non-Euclidean levels in a first person perspective. Players are limited to moving and jumping originally but eventually gain access to a strange gun which will again flip what you consider to be normal on its head.

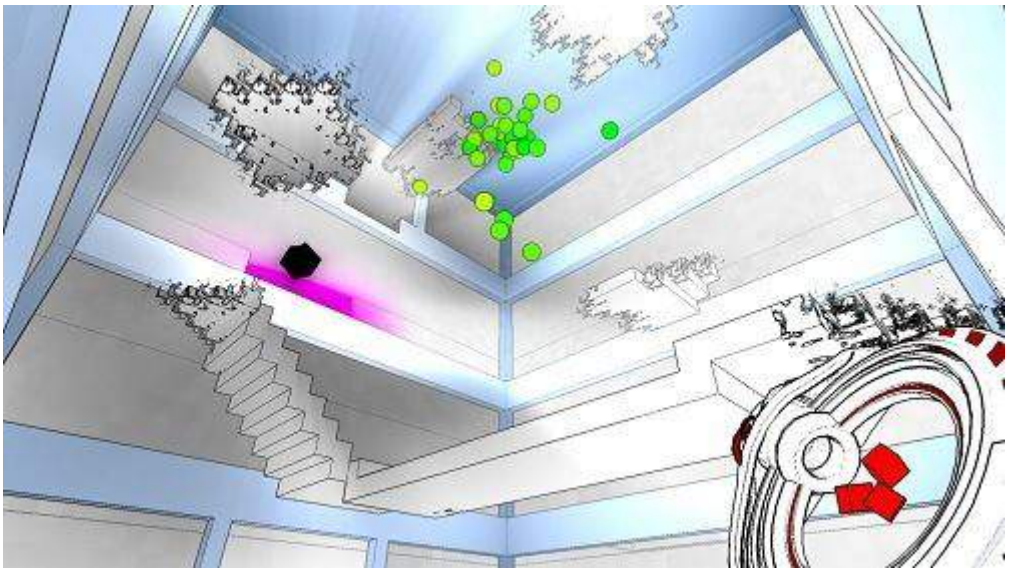


Fig 2. In-game photo: Antichamber

1.4. Main Aspects of the Game

Unity

In many cases game engines provide a suite of visual development tools in addition to reusable software components. These tools are generally provided in a rich IDE to enable simplified, rapid development of games. Unity is one of the most popular game engines. Like other types of game engines usually provide platform abstraction, allowing the same game to be run on various platforms. Unity provides an all-in-one editor offering easy access for the following components and features: 2D & 3D graphics, AI pathfinding tools, user interfaces, physics engines... Unity is a good choice not only because it has many features, but it is also well documented and there has been a good and supportive community developed around it. Unity supports two different scripting languages: C# and a JavaScript flavour language. Obviously C# is the best choice since it is more advanced than JS.

Character movement

Since, escaping from a maze is the goal of the game, implementing the character movement will be the first task. Fortunately a bunch of examples can be found on the internet, so this should not be the difficult part of the journey.

Scene creation

Creating the scenes, the stages, the parts of the actual maze will be an important task, because the game environment should produce a pleasant feeling in the player, since he probably will tackle the problem of maze for hours. If the maze will look like irritating and ugly, the player will not try to solve the puzzle. The puzzle itself could be frustrating enough and I do not want to give other reasons to the player to quit the game.

Fig 3. Kimnyoung Maze Park

Non-euclidean world

Implementing non-euclidean geometries in the game means that the shortest path between two points will no longer be the straight line. With such a game mechanics, the game can offer an interesting twist while the player tries to escape. Non-euclidean geometries are hard to explain, so here are two examples:

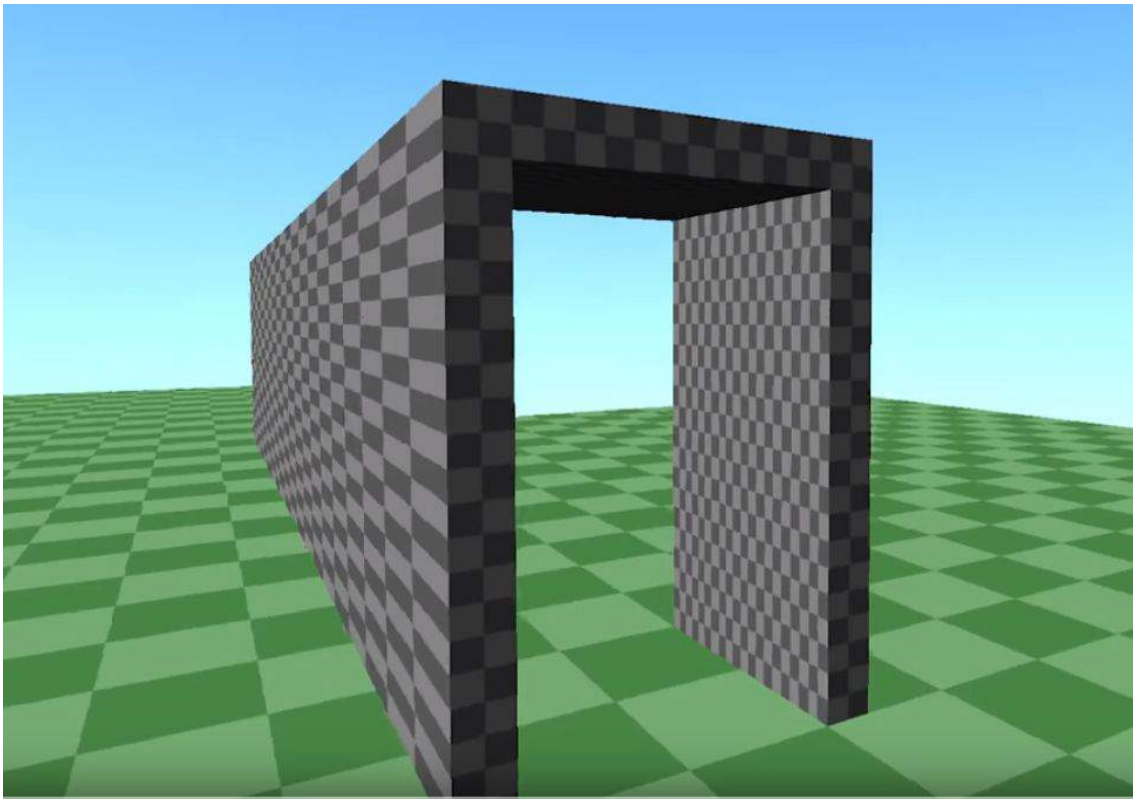


Fig 4. Tunnel which has a shorter interior, than its actual length that can be measured outside

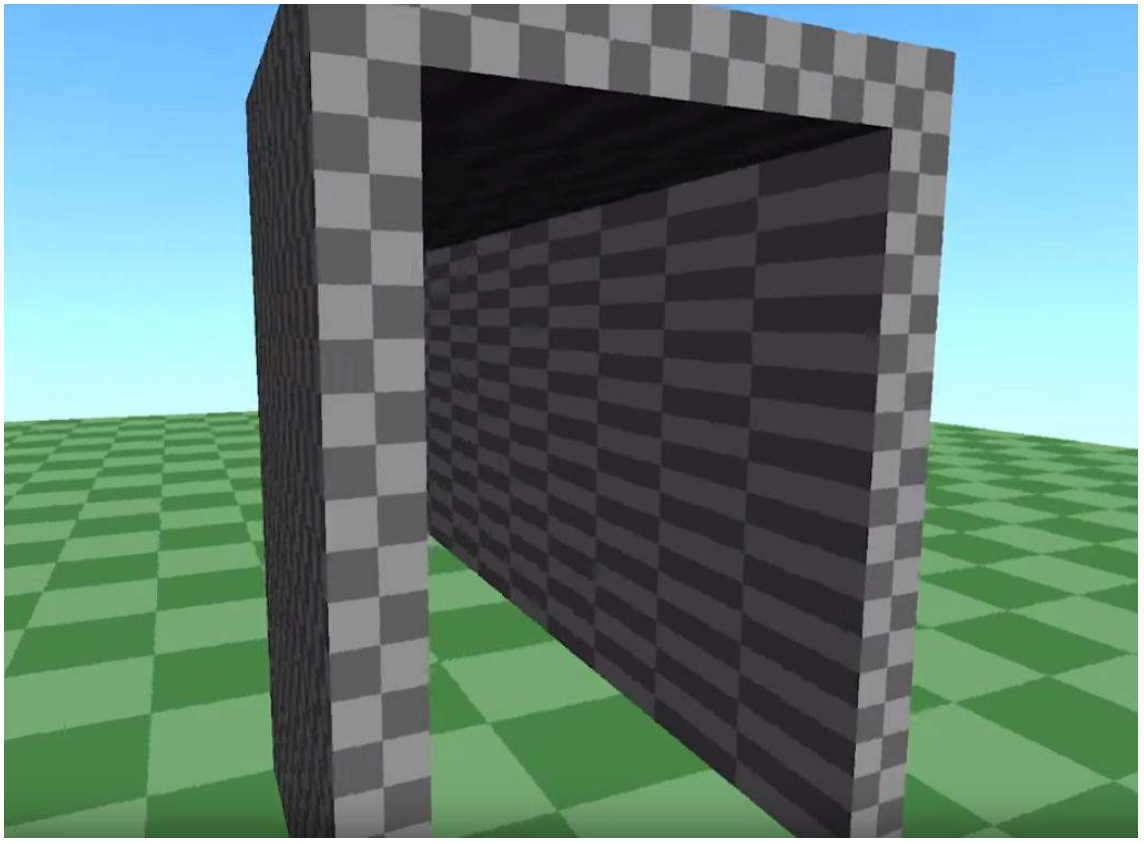


Fig 5. Tunnel which has a longer interior, than its actual length that can be measured outside

These illusions are created by the help of a pair of hidden planes, unique shaders, and camera transformations. Both, *Portal* and *Antichamber* use similar game mechanics to create illusions or interesting interactions with the game environment.
More example can be found here: https://www.youtube.com/watch?v=kEB11PQ9Eo8

The maze in my proposed game, will have these hidden planes ("portals") to create the illusion that the space is bending. Going through a corridor you can arrive to a junction that you just passed. The ability of orientation will not work in this maze in the common sense. The only certain elements of the game that the player can use for orientation will be the clues placed in each junction.

Achieving this kind of illusion is difficult, because will involve the following things:

- Hidden planes that serves as "portals"
- Custom shaders and rendering tricks for the above mentioned planes to create the visual illusion
- Object teleportation from one space to another, keeping its relative orientation and position
- Game camera teleportation which is similar to object teleportation

2. Analysis and Specification of Solution

In this project three people participates with the following roles:

Name	Role	Role description / responsibilities
Ervin Racz	Developer	Designing and developing the game. - Developing the game according to the roadmap / plan - Gathering feedbacks from evaluators / users - Must take into account the feedbacks during the game development
Sergiu Redeca	Evaluator	Prevent mistakes and defects in development phase to avoid problems when delivering solutions that can affect the "future" of the game. - Must provide feedback in each phase of the game - The evaluator should provide his feedback from the perspective of a typical user (from the targeted audience, see chapter 1.2.), game tester or QA tester.
Bogdan Ban	Evaluator	

This chapter is meant to be a more detailed description of the game and the game development process itself. Here we will discuss about solution proposals, roadmaps and tasks, and because, using the word "task" many times in the same chapter but with a little bit different meaning, can be very confusing, I would like to make distinction between tasks of the user (user stories) that can be performed within the game according to the gameplay and tasks of the developer and evaluator that are specified in a development roadmap. In the first place, we will discuss the tasks of the user (based on basic interactions between user and video game) in order to know what should be implemented by the developer and what has to be put in the roadmap.

Addressed to Evaluators

Are you familiar with the puzzle game genre?

Sergiu Redeca	Bogdan Ban
Yes	Yes.

If yes, what kind of puzzle game did you play?

Sergiu Redeca	Bogdan Ban
Tetris	Ballance, Neighbors from hell and a handful of other one time puzzles mainly focused on discovery and interaction of items.

Provide your comments on chapter 1.

Sergiu Redeca	Bogdan Ban
The article has a very good structure. The introduction might be useful for someone new in the game industry. The audience is well-defined and the idea of the game is interesting enough for starting to build a great game. The objectives of the assignment have been met.	This first part of the project is well written as it includes explanations and examples about the main mechanisms that will be featured, especially non-euclidean geometry**.** The link was particularly helpful in better understanding the concept.

2.2. Description and Analysis of Tasks

So, as I said we need to make a list of tasks (user stories) based on the gameplay and the basic interactions that can be made between user and game. These tasks can be performed within the video game with the help of a user interface or by the implemented game mechanics.

Nr.	Task (user stories)	Specifications	Opinions		
			Developer (Ervin)	Evaluator (Sergiu)	Evaluator (Bogdan)
1.	Navigating between game levels (labyrinths) with different difficulties using a game menu.	By clicking on the icon of the level, the player character will spawn in the maze.	The last position of the player could be saved when the user exits the game, and whenever the user enters the game again, the journey could continue from where he stopped playing.	Ok	Ok.
		After the spawn, a timer will start in order to count the time spent in the actual maze.	Timer could be saved also, just like the position of the player. This is not necessary, but at the end would be nice to visualize how much time did the user spent (in total) in the maze.	There can also be an hourglass to show how much time is left until the game ends.	An important element such as time which gives feedback to the player is a good idea.
		Until the tutorial is not completed, the player cannot play on “real” game levels.	It is important, to get familiar with the core game mechanics, later the game could introduce other mechanics...	Ok	Ok. Although some experienced players could argue that its wasted time.

		After completing the tutorial, the user would be able to choose whatever level he wants to play.	Different levels could contain various game mechanics and different type of clues and patterns that can lead the player through the maze.	The difficulty of the levels should be progressive. The player should not be allowed to skip levels.	Ok.
2.	Maneuvering with the game character	Using the standard key bindings for games (w,a,s,d), the user should be able to move with the game character forward, backward and aside.	This is essential, since the goal is to get out of the maze.	Ok	Ok.
		Jumping with space bar.	This feature is not so necessary.	Ok	Ok.
		Maneuvering around the space, from one junction to another according to the non-euclidean aspects of the game.	This should be as smooth as possible in order to create perfect illusions.	Ok	Ok.
3.	Completing the level	Completing a level should bring the user to the level selection screen.	Completing a level could be achieved by entering a nice, visually pleasant chamber, room or open area.	It would be better to show the user a menu to ask them if they want to go back to the level selection screen if if they want to go forward to the next level. Also, you can show a score.	Ok.
4.	Interacting with gameobject in order to solve clues	Each junction will have a number, and that number will be hidden from the player. Some kind of	Different kind of interaction and clues = increased level of curiosity.	Ok	An enriched environment of objects is important to keep the

		interaction with the gameobject must be necessary to reveal the number of the actual junction. The goal of the player will be to guess these numbers and then to use them to solve the maze.	Implementing dozens of different game mechanics to provide interesting interaction with game objects is time consuming, so most probably I will choose only one or two.		player of feeling repetitivenes s. But the rooms should not be too busy as well. Also, changing the colour or size of existing objects helps.
5.	Accessing game and gameplay options	Turn on/off music.	Requires some game assets and ui implementation that is independent from gameplay - should not be difficult.	Ok	Ok.
		Adjusting volume level.	- the same -	Ok	Ok.
		Regenerating a give maze from a specific game level.	This feature requires an algorithm that generates labyrinths based on some predefined rules. This could be difficult, especially if we have multiple game levels with different game mechanics.	Ok	Being a complex idea while the main game design already has hard features to implement, I recommend having a low priority for this.

2.3. Sketching the Prototype

A maze must be huge, so we want it to be procedurally generated. The procedural generation of maze imply the use of modular components of the maze. These modules will be the building block of the maze. My first thought was to create two simple junctions like on the following figure; they are easy to model in Blender or 3DS max or they can just be created by primitive shapes in the Unity IDE. Whatever we choose these modules must be prefabs in unity.

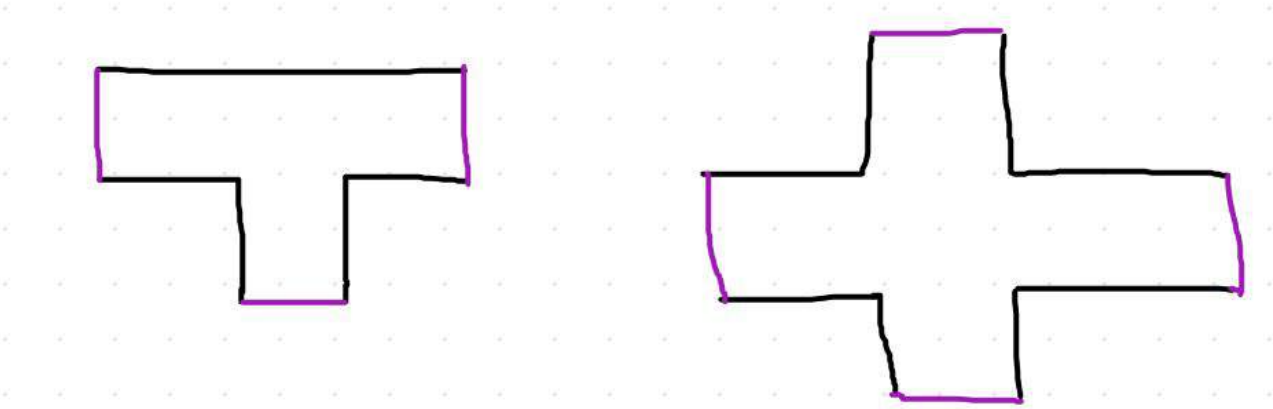


Fig 6. Two modules / junctions; first with three corridor, second with four

Giving a second thought to the shape of these junctions, we might see that we can make them more functional with some little changes. In this case we draw their shape by starting from circular room, as well as a square one. This way we have more space for the clues / game objects to be arranged.

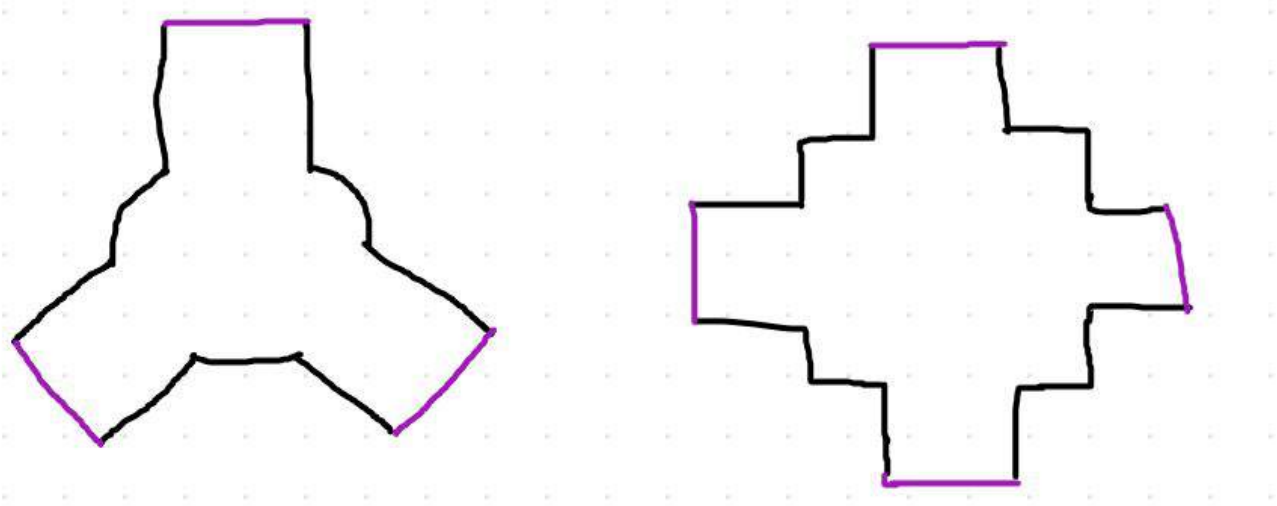


Fig 7. Junctions with more functional space

So, if we choose the first type of module from fig 7., which is like a triangle, then the maze would look like on the fig 8. It is important to note that each junction will have a unique number, because these numbers will lead the player out of the maze. These numbers will be hidden in corresponding junctions and the users will need to guess them. The clues, rebuses, mathematical problems placed in

the junction will help the user to make the correct guess. Also a graffiti mechanics would be nice to be implemented to write notes on the walls.

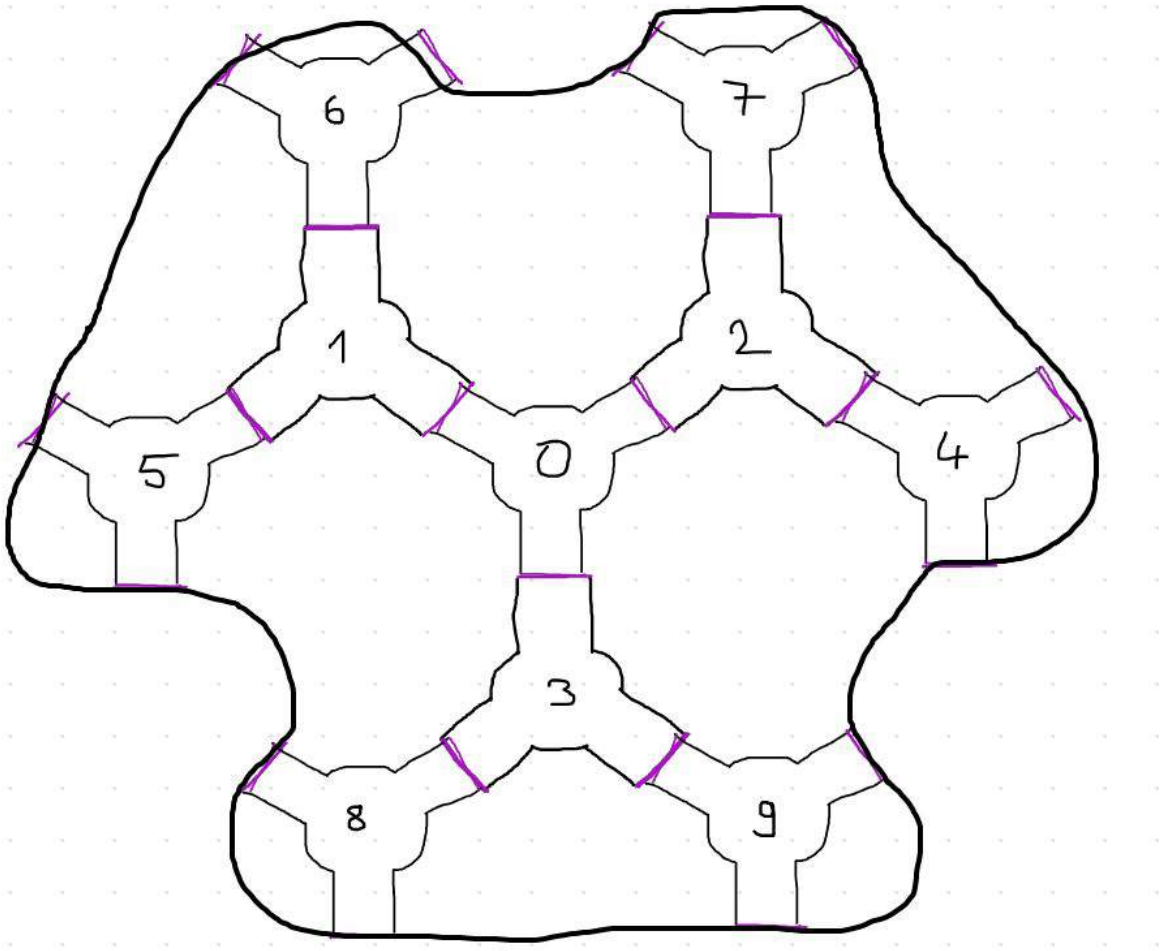


Fig 8. Maze built from modules - each junction has a unique number

For example, the maze could have the following solutions:

- Player spawns at junction 0 and should follow the even numbers (0 ---> 2 ---> 4 ...)
- Player spawns at junction 0 and should follow always the bigger numbers (0 ---> 3 ---> 9 ...)
- The player should always follow the prime numbers
- The player should follow the numbers that can be divided by a number x.

It is not difficult to see, that the number of patterns and sequences that can be used for solutions is infinite, which is a big problem, especially if the maze is gigantic and the exit door is far away from the player. If the number of possible solutions are infinite and the player can move freely in the gigantic maze, then the chances are nearly zero for the player to find the right solution. Even if the player succeeds, that will be a fully heuristic success. Because of this, the game needs to offer some reference points, that the player can use to create patterns and make deductions…

Fig 9. Graffiti in Unity

So, speaking of lack of reference points, how does the player know if he has chosen the right path for his escape? Simply, he does not. This is a puzzle game… the player should be able to find the solution based on his inductive reasoning aptitude (pattern recognition aptitude) and deductive skills. That is why we need the non-euclidean aspect of the game, because those mechanics that were shown in *Antichamber* and *Portal* can help the player to create patterns. In this game I will use only the "bended space" trick which can be achieved by portals.

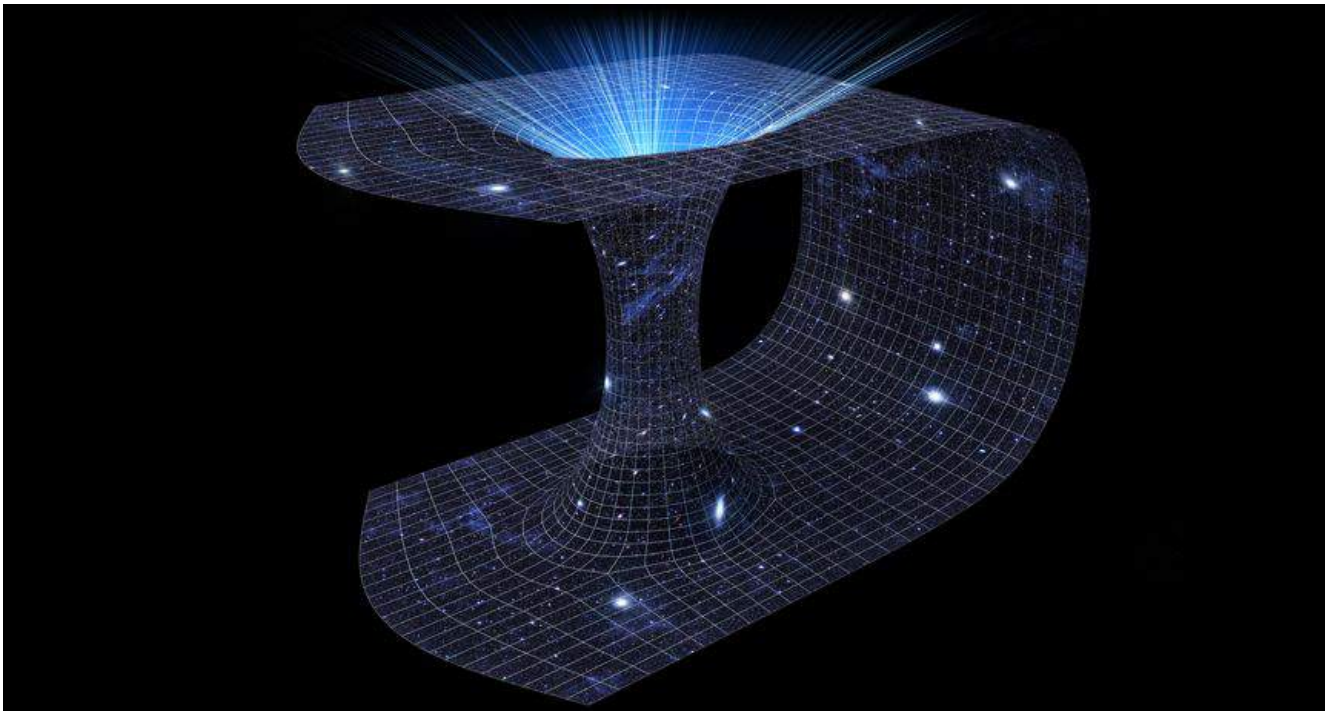


Fig 10. Wormhole, bended space

The world full of non-euclidean elements can be very confusing, because you except one thing that you are used to, but other things happen. For example, considering the following maze, the player starts to walk to west through junctions 2, 5, 14 and then he will end up in the east (see the red arrows in the figure are links between corridors). All of the wrongly chosen junctions must lead the player back to the starting point. This way if the player enters a junction that is familiar him, he can make the deduction, that he was wrong at the previous junction or before previous.

Note: the maze should be generated that way that, the player will get back to a familiar junction that is near the starting point (max two junction away) if continues to go along a wrong path. A wrong path should also not be longer than two the series of two junctions. This means that the starting point will be always at most 4 junctions away.

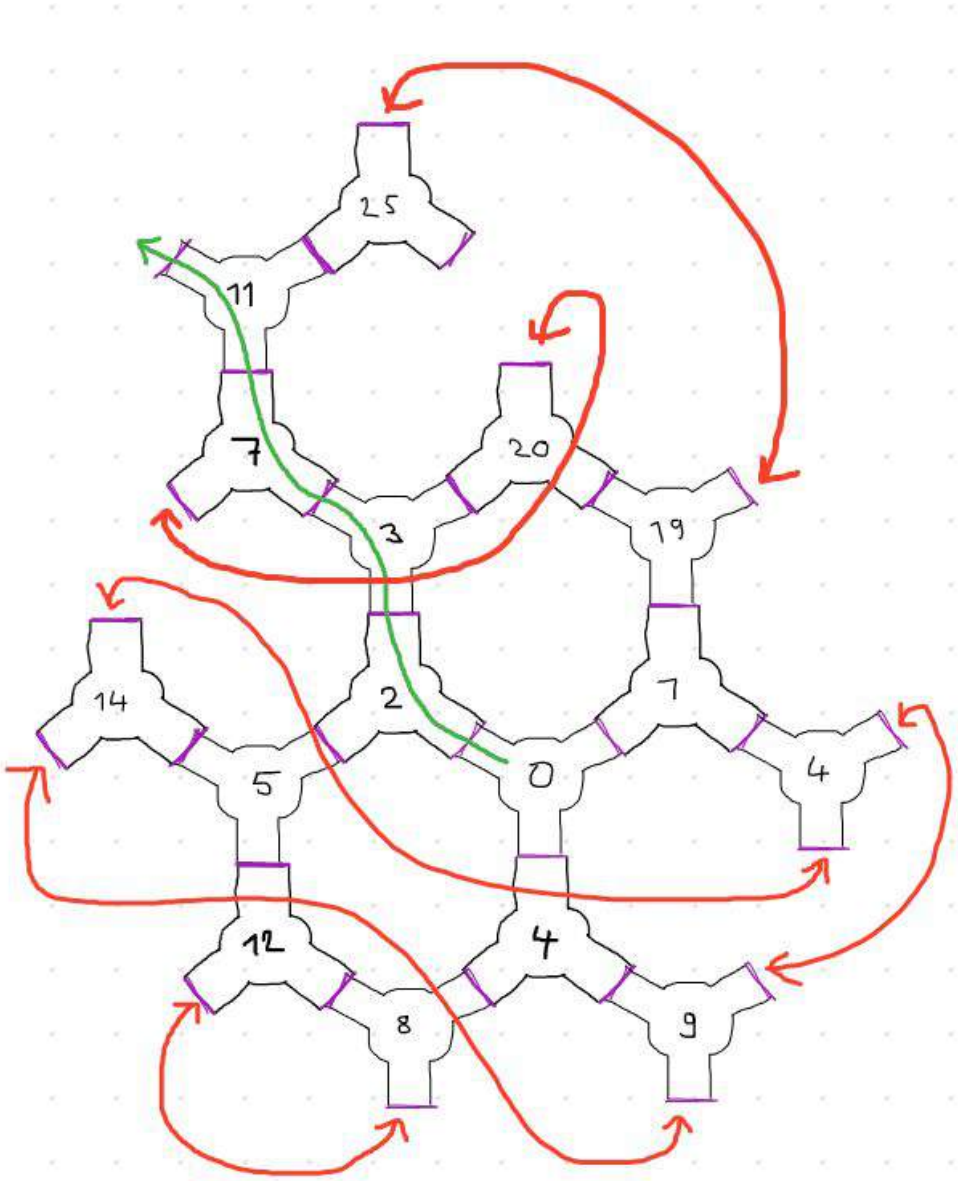


Fig 11. Maze with a non-euclidean twist; green is the good path; red arrows are links between the

corridors

Apparently the maze will look like in the fig 8. for the users, as a coherent mess of junctions physically connected together, but in order to facilitate the procedural generation of the maze, in the reality the modules will be completely separated and the illusion of coherence will be made with portals:

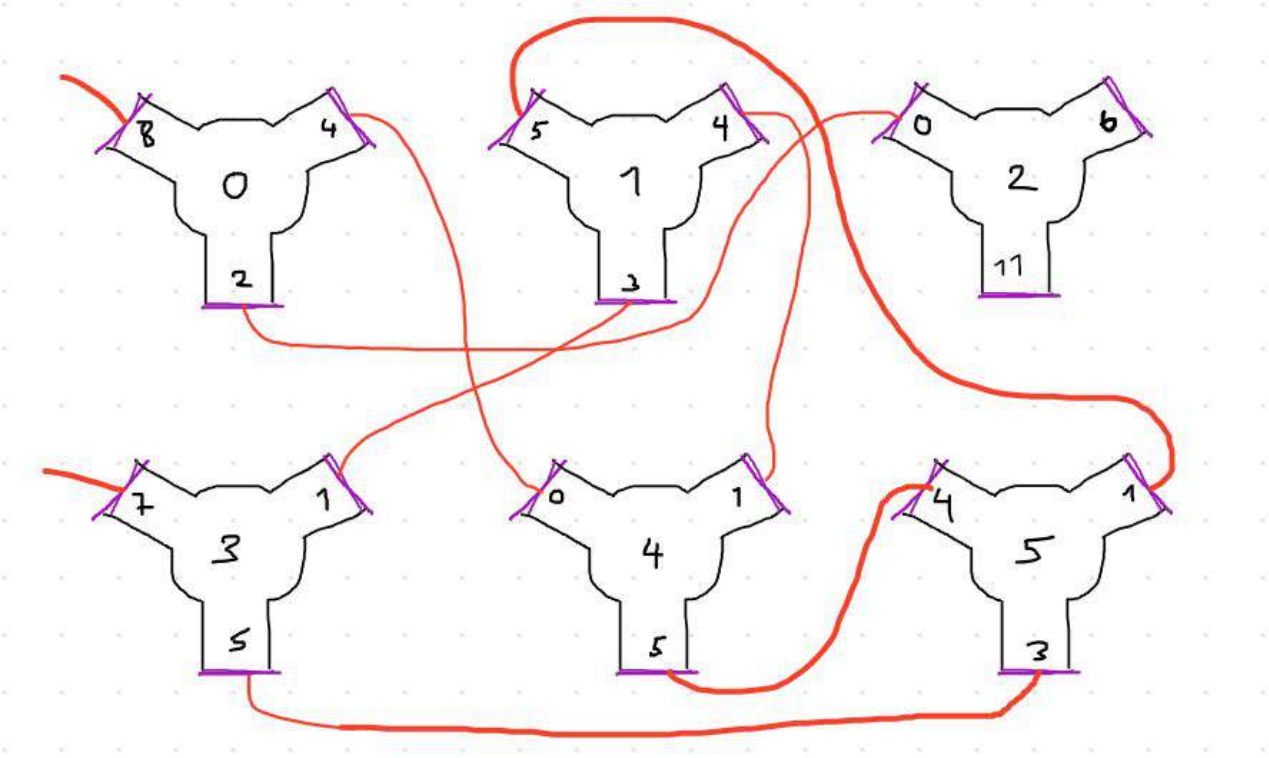


Fig 12. The maze will look like soldiers arranged in lines from the sky. The illusion of coherence for the user is achieved by “portals” (portals are the purple planes at the end of a each corridor, and links between two portal-plane are illustrated by red lines)

2.4. Roadmap

Epic	Task	Description	Deadline
3D Scene design	Design the building blocks of the maze	Should produce a 3d sketch wireframe, and the actual prefab in unity.	28 March - 10 April
	Create decorative elements for the maze	Decorative elements can be the part of the junctions, corridors. The purpose is to design and	28 March - 10 April

		create a visually pleasant environment.	
	Find a solution to represent numbers that are associated with junctions and make a prototype	In the first prototype this can be a simple wooden plate with the numbers drawn on it.	28 March - 10 April
	Extend the prefabs with the plane of the portal-gates	Source of Inspiration: https://www.youtube.com/watch?v=cuQao3hEKfs	28 March - 10 April
	Design and create a prototype for the finish	An open area, a nice chamber, a cool room, which represents the end of the maze. Entering this room, the game should unlock the next game level. The game should also suggest to proceed to the next level.	28 March - 10 April
	Gathering feedback from evaluators and QA		10 April - 12 April
Maze generation	Create the first working portal	Create the link between two junction, making the illusion that they are physically connected. https://www.youtube.com/watch?v=cuQao3hEKfs	28 March - 10 April
	Design the maze generation algorithm	Define the general rules for the algorithm, then use graph theory to be able to generate the abstract representation of the maze.	8 April - 15 April
	Investigate and design a UML diagram for the procedural generation of a maze	UML diagrams: https://www.smartdraw.com/uml-diagram/ Maze generation should be customizable by parameters. Possible solutions for the maze generation interface: ● https://docs.unity3d.com/Manual/editor-EditorWindows.html ● https://docs.unity3d.com/ScriptReference/Edit	8 April - 15 April

		or.html • https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.html	
	Testing / QA and Gathering feedback from evaluators for the second phase of the project	Test the character movement and the first working portal	15 April - 17 April
	Implement the maze generation algorithm		19 April - 30 April
User interactions	Implement character movement	Using the standard key bindings for games (w,a,s,d), the user should be able to move with the game character forward, backward and aside. Optional: SPACE bar for sprinting and SHIFT for sprinting	8 April - 15 April
	Implement ability to draw on walls	The goal is something like this: https://www.youtube.com/watch?v=M71K1bkadsc	19 April - 30 April
	Save game state on exit	Exiting the game, the states should be saved (player position and orientation and the spent time on each level)	30 April - 8 Mai
User interface	Design game menu	**Navigation between levels**: • After completing the tutorial, the user would be able to choose whatever level he wants to play. The menu should be accessible from everywhere. **Settings menu** should provide option: • to adjust master volume, turn on/off the music • to reset levels which	30 April - 8 Mai

		means the deletion of the player data from the respective level (pos, orientation and spent time) + regeneration of the maze.	
	Implement the menu		30 April - 8 Mai
Level design	Design new game mechanics for clues and puzzle solving which has great synergy with the core game aspects	Designing more advanced game mechanics for clues and puzzle solving. Sergiu - “The difficulty of the levels should be progressive. The player should not be allowed to skip levels.”	Not specified yet
	Extend the maze generation algorithm to be even more customizable	Having a more customizable maze generation algorithm, with an easy to use editor interface, designing new levels might be much more simple and effortless.	Not specified yet
	Design game levels	In the prototype, the first levels will use only the core game mechanics (basic character movements, non-euclidean junctions, basic maze generation algorithm, graffiti on wall), but higher levels may use other, much more advanced game mechanics.	

<u>Addressed to Evaluators</u>

Is the idea of non-euclidean world understandable?

Sergiu Redeca	Bogdan Ban

Yes, it is.	Yes, especially with the link of reference providing even more examples.

Provide your comments on chapter 2

Sergiu Redeca	Bogdan Ban
Very good, in-depth specification.	The chosen triangle type junctions instead of the square ones is a nice choice because the triangle room has a different atmosphere in comparison, as it looks better when they are combined, however the square one would look like a giant plain infinite room. The explanation of how the illusion of never ending rooms works is very good and the figures illustrate it clearly. The roadmap is also thoroughly described.

Developers reply to the evaluators opinion from the table of user stories / tasks, in the subchapter 2.2.:

Sergiu -
"There can also be an hourglass to show how much time is left until the game ends."
Ervin -
I will investigate how to create a hourglass in unity. In my opinion, implementing a 3D hourglass will not be an easy task. This probably will be low priority for me.

Sergiu -
"The difficulty of the levels should be progressive. The player should not be allowed to skip levels."
Ervin -
Good point. Completely agree on this. I added a new epic to the roadmap, right after reading this comment (see Level design).

Sergiu -
"It would be better to show the user a menu to ask them if they want to go back to the level selection screen or if they want to go forward to the next level. Also, you can show a score."

Ervin -
I like this idea. Most probably will be implemented like you suggested. The score in our case will be the time of completion of the level.

---

3. Designing the Game

Several user stories, game mechanics were discussed in the previous chapter, as well as the core game strategy. It was only a broad discussion, so in this chapter - solidifying my prospects - I want to give you an insight about the gameplay with some details.

3.1. Game Scenes and 3D Objects

I chose to make junctions with three corridors, because of the following reasons:

- The network of this type of node, results in a smaller graph, which is more manageable than a graph made of nodes with four or even more edges
- The total number of corridors in a maze must be odd, because one of them should represent the goal of the player (the end of the maze)
- Having triangular modules provides the possibility to display clues, texts, numbers on the wall for the player in the front of player

In this version of the game only to types of static objects will be created. The triangular junctions and the clues, which will be placed on the wall in the form of text. The scene will be generated procedurally by hitting the proper button from the game menu.

The most important dynamic objects will be the portals, that will be made of simple planes with a custom shader. They can be called dynamic, because the player can interact with them, by colliding with them or moving around the space (they change their texture in real time based on player's camera movement and then they transform the player's position into another junction if it hits the plane of the portal).

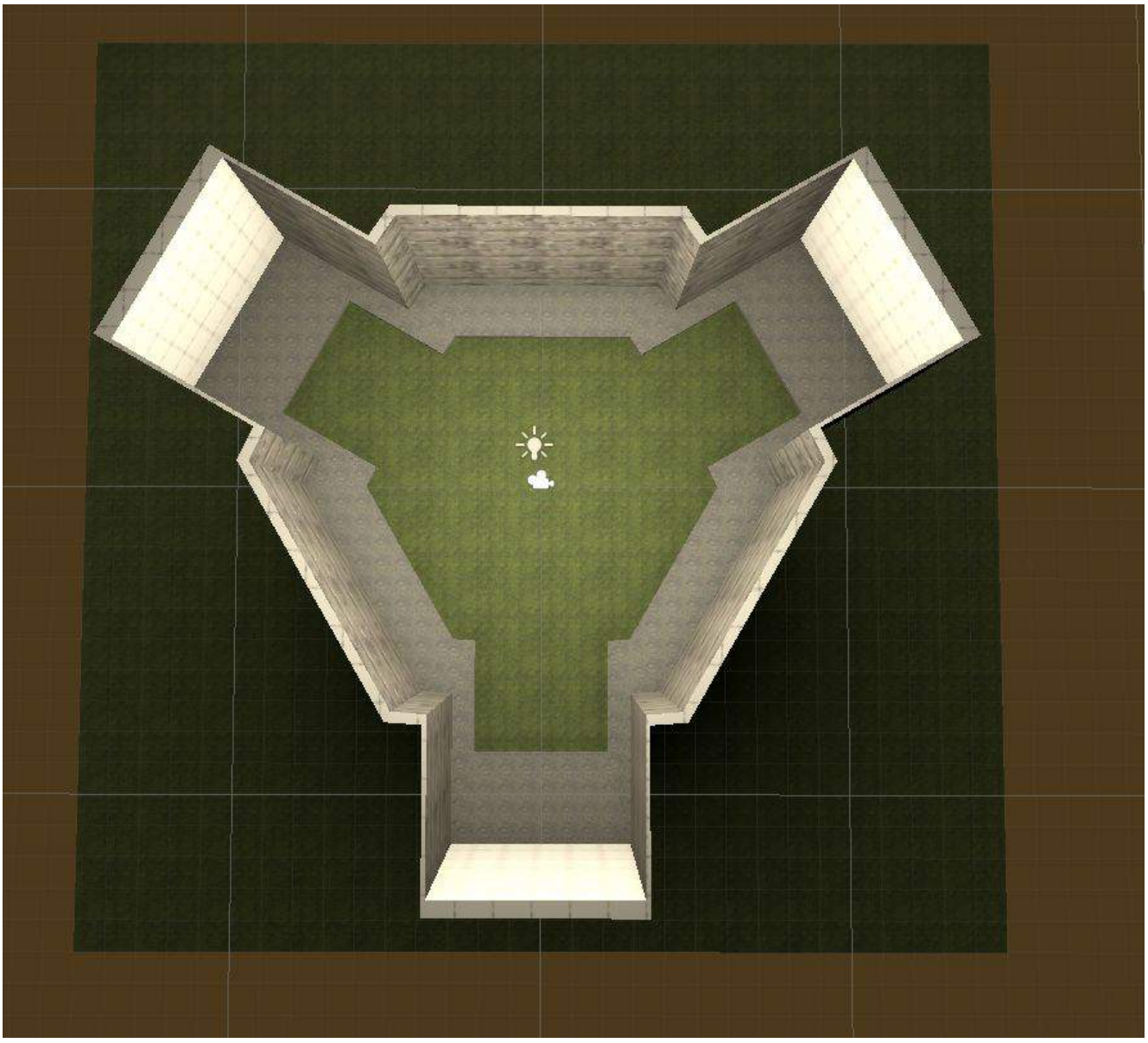


Fig 13. Junction with three corridors - symmetrical point light

Having static, baked in lights into the static objects is a good trick to reduce the number of required computation for the GPU. In this case the light rays come from a point-light-source which is placed exactly at the center of the junction. Each junction will have its own light-source. Asymmetric light-sources could cause strange phenomenon with the luminosity near the portal planes, on the wall.

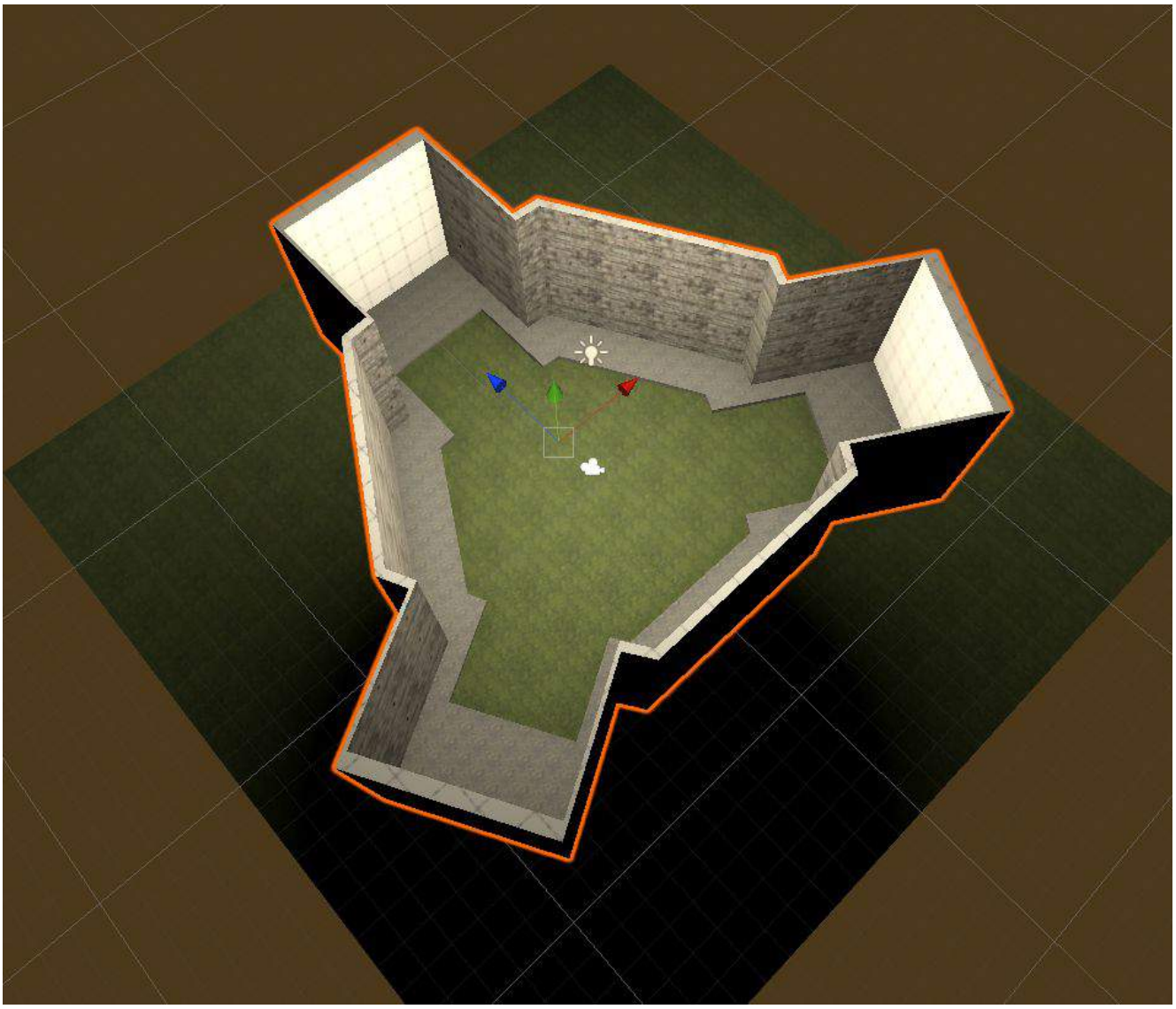


Fig 14. Junction made with ProBuilder

I chose to model the game objects with the ProBuilder plugin.

“ProBuilder is a unique hybrid of 3D modeling and level design tools, optimized for building simple geometry but capable of detailed editing and UV unwrapping as needed.

These tools let you quickly prototype structures, complex terrain features, vehicles and weapons, or make custom collision geometry, trigger zones, and nav meshes.”

This way I was able to simply create a prefab within the Unity Editor without the need of third party softwares like Blender, 3ds Max, etc.

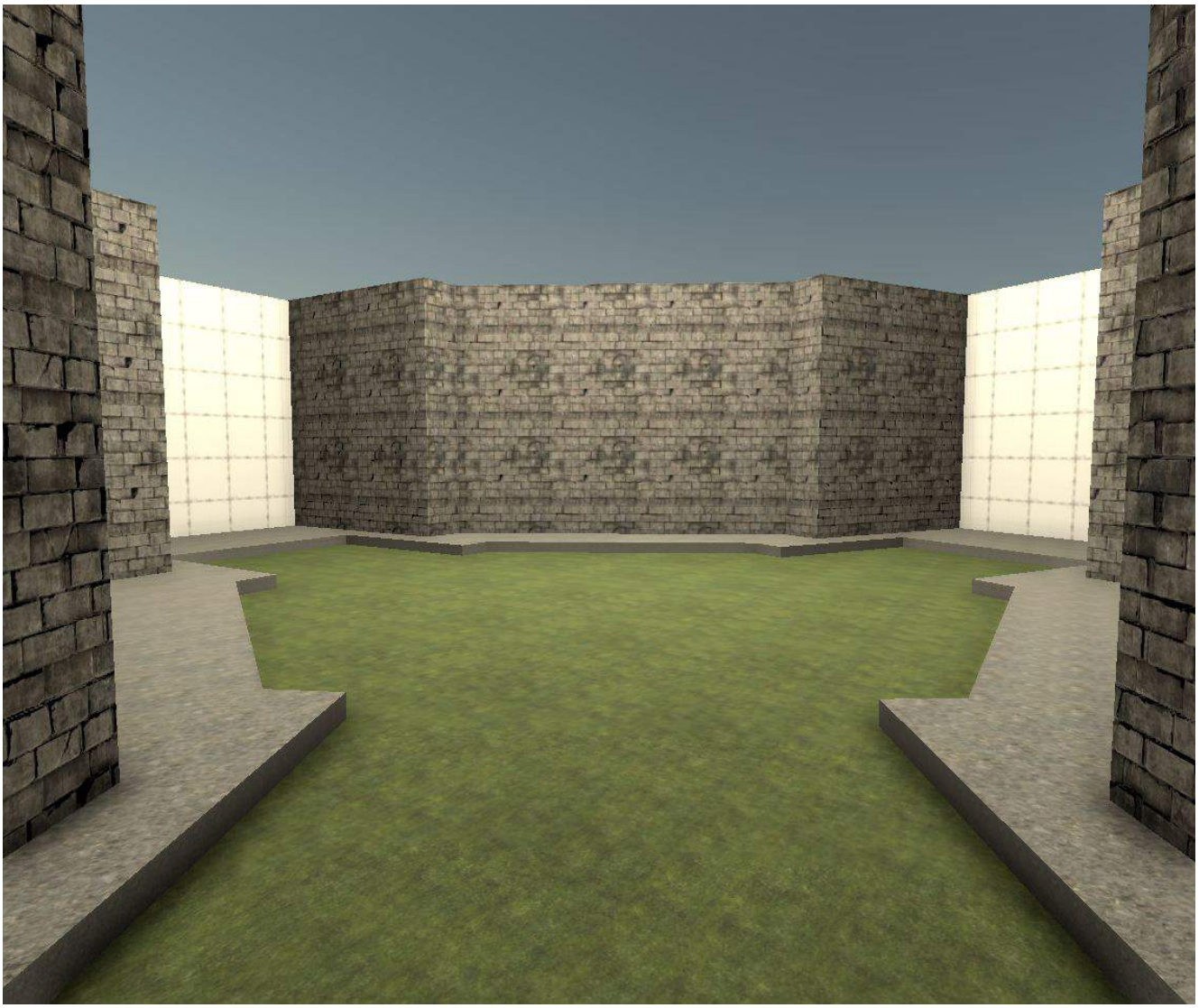


Fig 15. Inside a junction. Portal planes will be placed on the top of the white walls

After spending some time with changing the look of the junction, the furnished room has the following appearance (see **fig 16**). My intention was to make the scene look like an abandoned maze which is frightening as much as possible. With respect to the previous look, the new junctions has the following assets in addition: curtains, banch, towers, lamps and numbers (3D text with custom font-shader, inspired by this tutorial). These assets were downloaded from the unity asset store (Cartoon Castle Building Kit). I placed 7 point lights with more yellowish color emission, which brings the feel of desperation to the place. The feeling could have been intensified with audio effects, but unfortunately I did not find any free audio materials that I liked.

Moreover, the maze will have more than a dozen junctions and, since customizing each junction on each level would be very time consuming, I did create variation of junctions and instead of making custom clues for each junction (room), I put the numbers on the walls. This means that the numbers on the walls are the only things that the player can heavily rely on in order to escape.

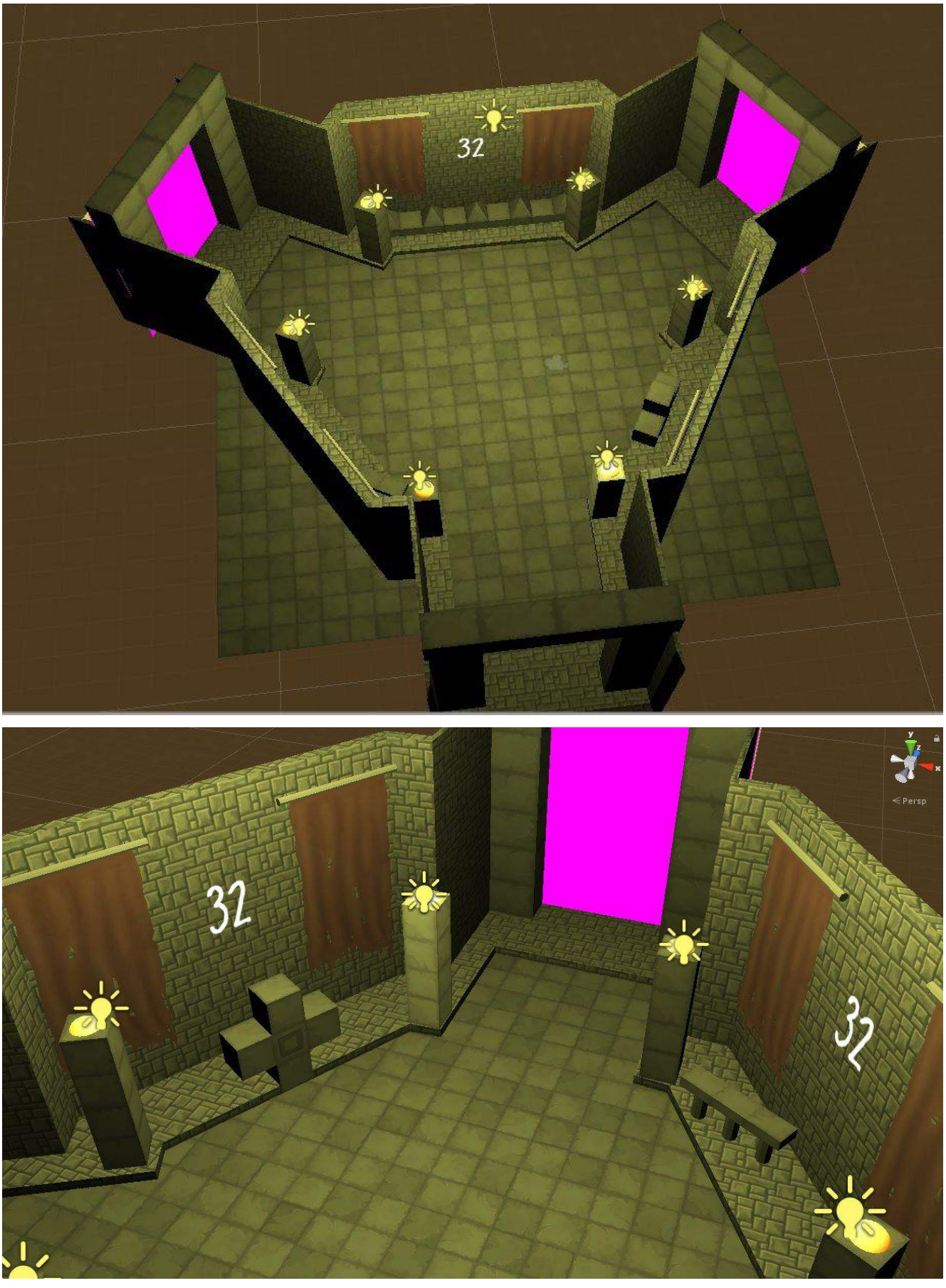


Fig 16. Furnished junction - building block of the maze

3.2. Game Strategy

The objective of the player is simple, and it is to find the end of maze. The player will spawn in the middle of a maze and his only hope will be to find the connections between the viable path and the clues placed in junctions. A time will be placed in the right-top corner of the screen az a GUI Text which will indicate the remaining time for the player to exit the maze. The timer will be counting down and if the player cannot find the end of the maze within the available time-frame, the journey will start over, by regenerating the maze with a new random structure.

[Player score for a particular level] = 1 / ([number of maze regeneration] * [timer initial value on the level] + [timer value when level is completed]) * 1000

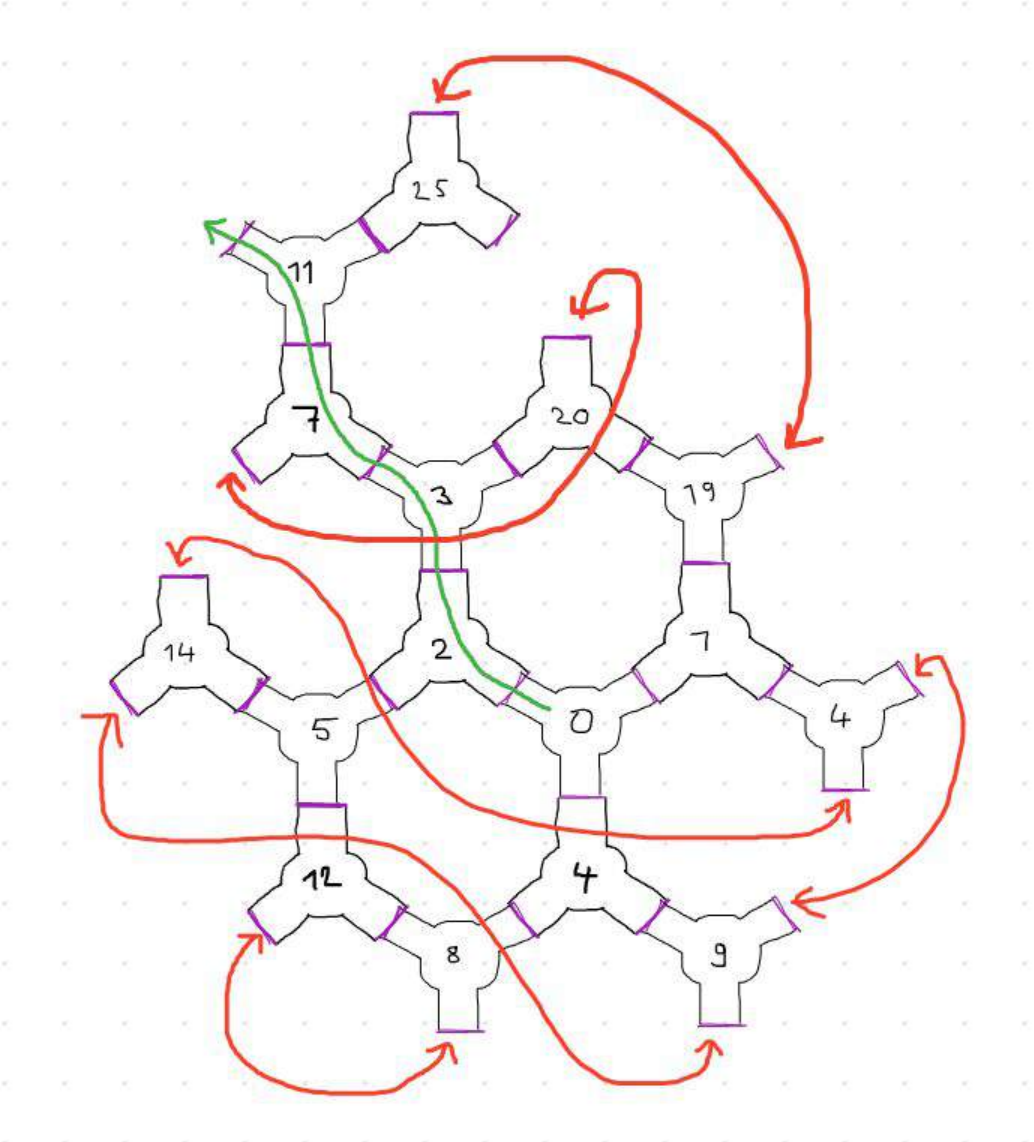


Fig 17. Correlation

3.3. Interaction with the Scenery

The main interactions within the game are the movements and the portal. The first interaction is obvious, with a pop-up message, the player can be thought how to move in the game very easily, but the second interaction is a tricky one, because its purpose is to create an illusion that portals does not exists.

Here are some thoughts how to achieve this illusion:

1) Rendering: Create a custom reflection shader. Add two planes and additional cameras to the scene. Project the view of the cameras onto the planes with the help of the custom shader. Modified camera transformation matrix knowing the relations between 2 portals and the player camera.

2) Object teleporting: Add inner trigger zone to portal which activates when moving object hits the collider. The trigger can simply create a clone of this object and add script to it, that controls his position and rotation using similar transformation matrix (like in previous step).

The technique is discussed in details in the following video:
https://www.youtube.com/watch?v=cuQao3hEKfs

4. Implementation

This chapter presents the Unity technology as well as parts of the implemented game in this engine.

4.1. Unity Technology

Unity engine is a software that provides game and simulation creators with the necessary set of features to build games or other applications, that require 3D or 2D graphical visualization, quickly and efficiently. Unity engine is more than a framework, it is an ecosystem for game / simulation development that supports and brings together several critical areas. You can import art and assets, 2D and 3D, from other software, such as Maya, Blender, SolidWorks, 3ds Max or Photoshop; assemble those assets into scenes and environments; add lighting, audio, special effects, physics and animation, interactivity, program logic; and edit, debug and optimize the content for your target platforms.

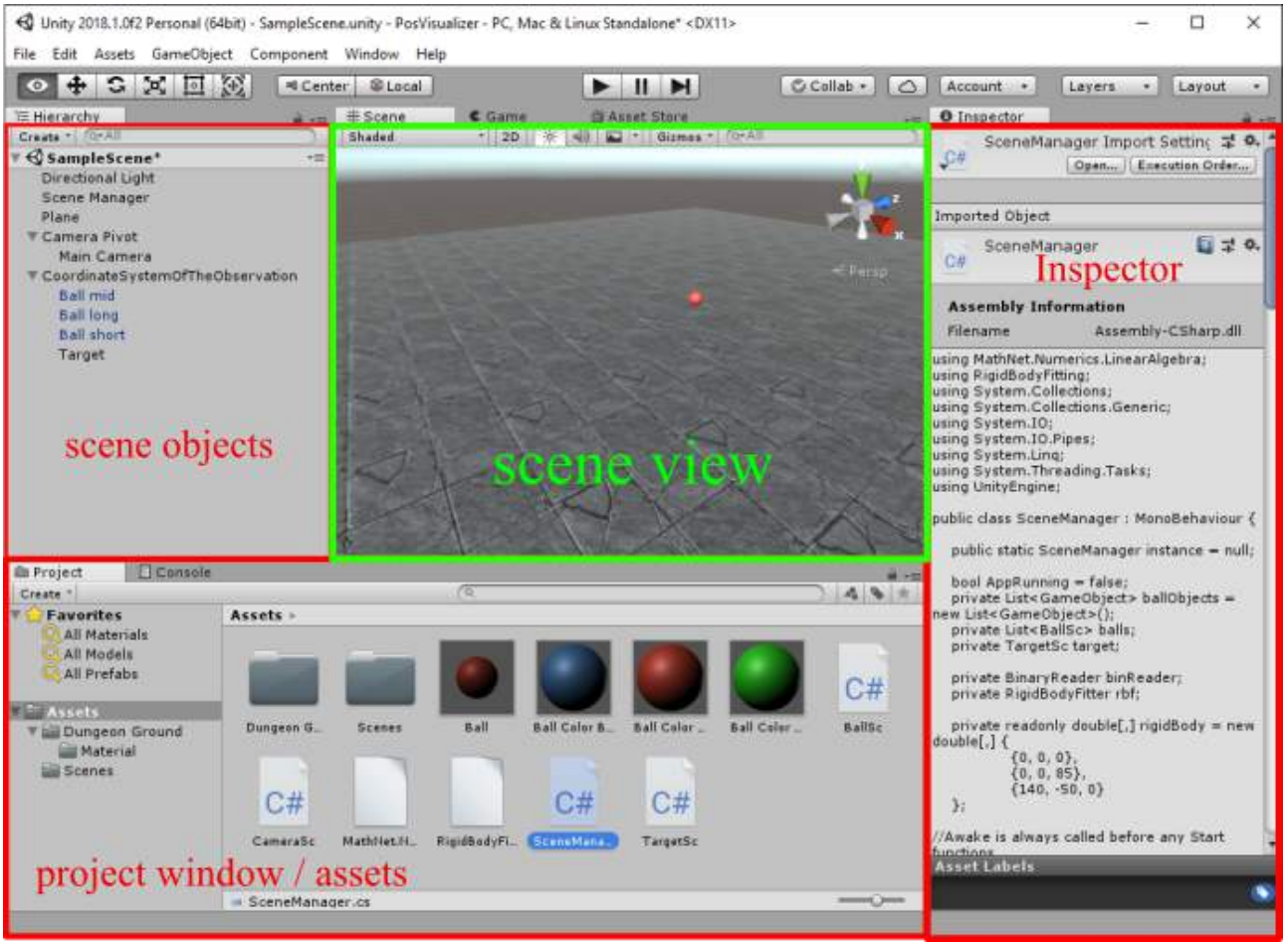


Fig 18. Unity editor

The user interface of the Unity editor is fairly intuitive. The scene view shows the objects that make up the game. So, objects in the scene are called game objects. At the moment (on **fig 18**), there are more than one game object in the scene that are visible: a camera, a plane (ground), three balls. These scene objects can be seen in the scene view and also in the hierarchy on the left panel. If you click on

one, Unity shows you information about that object in the inspector (see **fig 18**). The inspector shows all the different components that make up the game object or any asset. All game objects have a transform which holds information about its placement in the scene and its scale. For example, a ball object must also have a mesh and material component to be visible. Via the inspector, many other properties can be set to the game object, such as lighting or behavior properties.

An asset is representation of any item that can be used in your game or project. An asset may come from a file created outside of Unity, such as a 3D model, a DLL program file, an audio file, an image, or any of the other types of file that Unity supports. There are also some asset types that can be created within Unity, such as an Animation Controller.

4.2. Game Implementation

https://github.com/ErvinRacz/maze-of-numbers/

4.2.1 Character Movement

The resources that makes the FPS movement possible in this game can be found in my github repository. It is quiet a simple implementation, since it consists of two scripts and two game objects (Player, PlayerCamera, PlayerMove.cs and PlayerLook.cs). There is the Player object which has a character controller component and the script attached to this game objects is responsible only for the translational movements.

The following code-snippet presents the key function of the above mentions script. The constants and the input controllers are assigned in the Unity editor (see **fig 19**).

```
    private void PlayerMovement()
    {
        float vertInput = Input.GetAxis(verticalInputName);
        float horizInput = Input.GetAxis(horizontalInputName);

        Vector3 forwardMovement = transform.forward * vertInput;
        Vector3 rightMovement = transform.right * horizInput;

characterController.SimpleMove(Vector3.ClampMagnitude(forwardMovement +
rightMovement, 1.0f) * movementSpeed);
        SetMovementSpeed();
```

```
        JumpInput();
    }
```

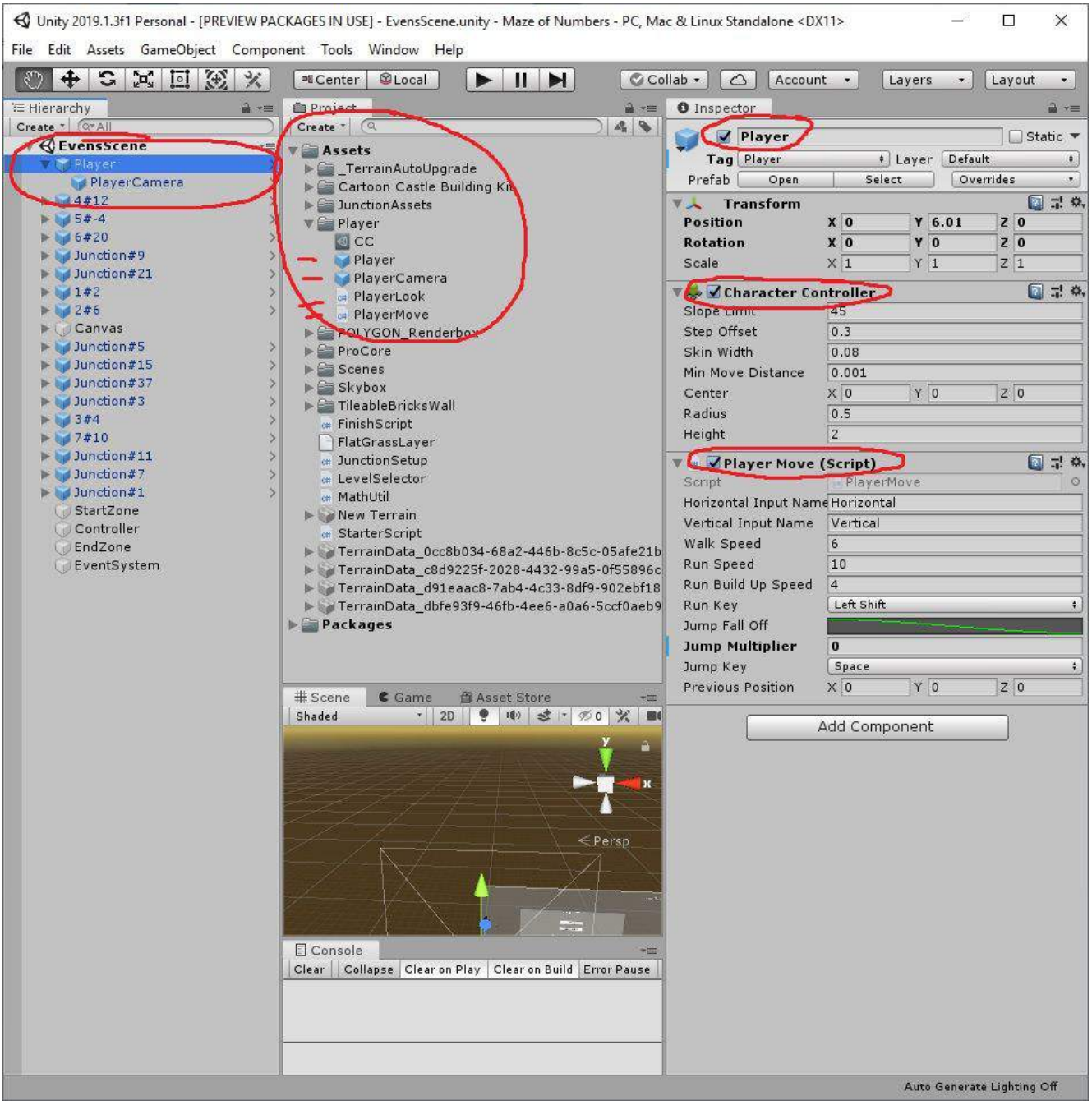


Fig 19. Player prefab

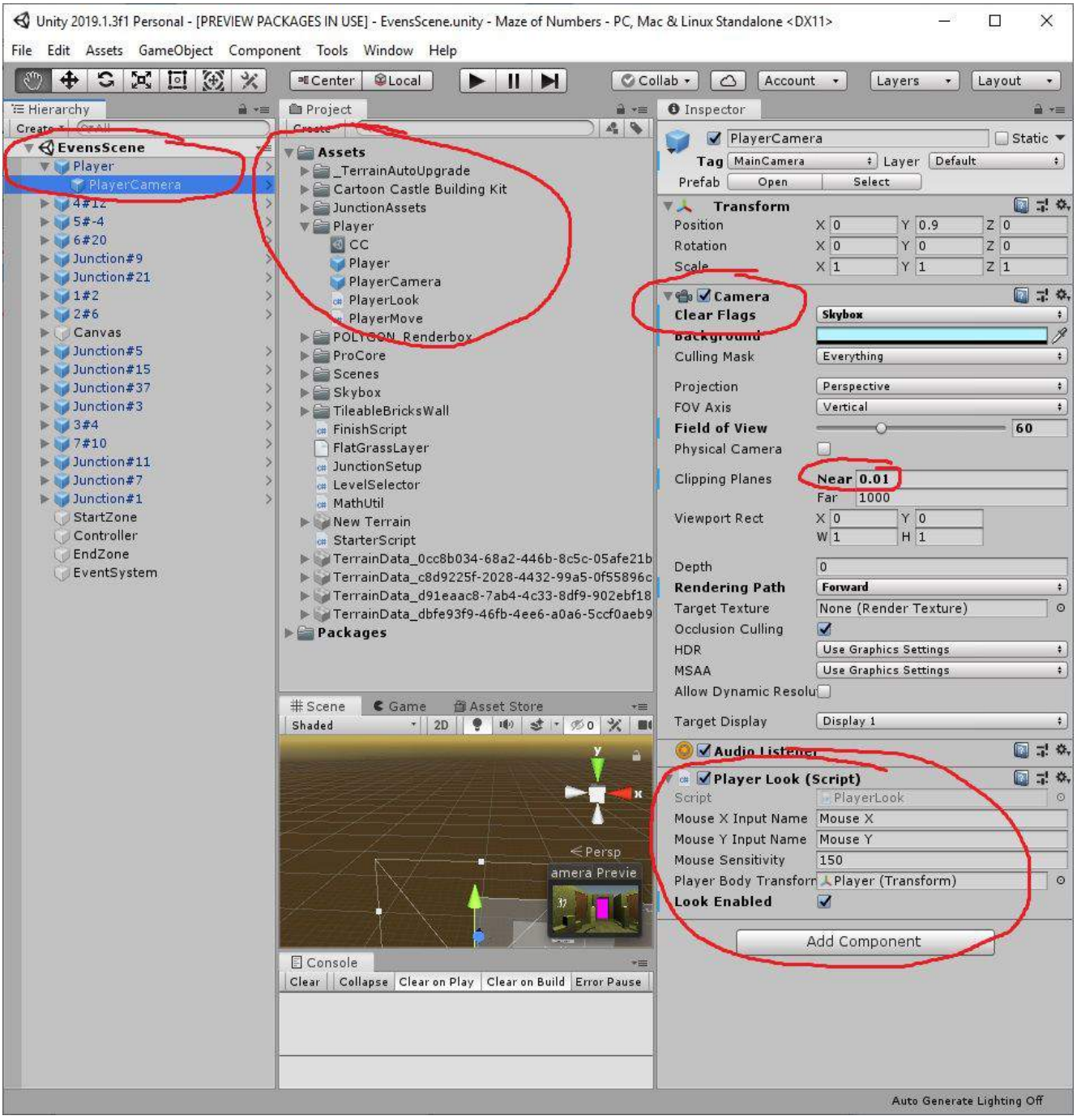


Fig 20. PlayerCamera prefab

And finally there is the PlayerCamera object with the PlayerLook.cs script, which are responsible for setting the player-look-orientation. The player camera is controlled by the mouse inputs.

Again, the most important function, which contains the logic of the camera movement, is presented in the following code snippet. The ClampXAxisRotationToValue(float deg) function is responsible for not letting the camera to turn over (upside-down) when the player moves its mouse in the upward or downward directions with excess.

```
private void CameraRotation()
{
    float mouseX = Input.GetAxis(mouseXInputName) * mouseSensitivity *
Time.deltaTime;
    float mouseY = Input.GetAxis(mouseYInputName) * mouseSensitivity *
Time.deltaTime;

    xAxisClamp += mouseY;

    if (xAxisClamp > 90.0f)
    {
        xAxisClamp = 90.0f;
        mouseY = 0.0f;
        // Stops the camera rotation when the player looks up
        ClampXAxisRotationToValue(270.0f);
    }
    else if (xAxisClamp < -90.0f)
    {
        xAxisClamp = -90.0f;
        mouseY = 0.0f;
        // Stops the camera rotation when the player looks down
        ClampXAxisRotationToValue(90.0f);
    }
    // To look up and down:
    transform.Rotate(Vector3.left * mouseY);
    // To look left and right:
    playerBodyTransform.Rotate(Vector3.up * mouseX);
}
```

4.2.2 Numbers on the Wall

As I mentioned earlier, in this version of the game, having only few hours per week working on this project, I did not achieve to have individually customized junctions in the maze. Junctions are the building blocks of the game, so creating a good parameterizable core game element was very important. To achieve this, I decided to use prefabs everywhere where the Unity framework let me. So, as the result of this architecture (provided by prefabs), I can place the default junctions in the scene by simple drag-n-drop interaction with the Unity editor. In the future, in order to customize the junctions that were placed in the scene, we can use the regular unity scene editing tools or the more advanced ones like the ProBuilder. Another option to make distinction between these junctions is to give them a name or an index number which uniquely identifies them. Since this version of the game is based on numbers, as its name suggests that, you will see only numbers on the walls (see **fig 21**).

The implementation consists of a unique text shader (custom text-shader, from this tutorial) and a script (JunctionSetup.cs) which takes the name of the game object and extracts the string that can be found after the first hashtag (see **fig 22**).

Fig 21. Numbers on the wall

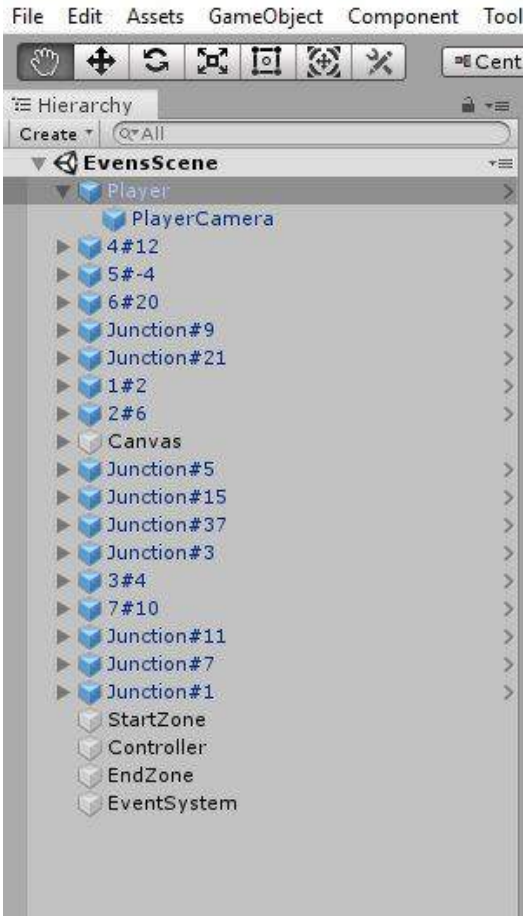


Fig 22. Junctions` name

4.2.3 Portals

The implementation of the portals are quite complex and I do not want to lie, I have struggled a lot with this section of the game. This documentation is not intended to provide a tutorial for creating such game mechanics, if it was so, then I would need to produce a lots of figures, diagrams and visual materials to explain how things work. Instead of that I would only point towards some online learning materials that can be good starting points to understand my implementation.

Learnopengl.com is a good source for learning, and the following pages are the most relevant in our case:

- https://learnopengl.com/Getting-started/Transformations
- https://learnopengl.com/Getting-started/Coordinate-Systems
- https://learnopengl.com/Getting-started/Camera
- https://learnopengl.com/Getting-started/Shaders
- https://learnopengl.com/Getting-started/Textures

The most important objects in scenes are the portals. They are made of simple planes with a custom shader. Players can interact with them, by colliding with them and moving around the space (they change their texture in real time based on player's camera movement and then they transform the player's position into another junction if it hits the plane of the portal).

A good tutorial can be found on this topic, which shows you a simple way to combine these game elements with the aim of creating a portal.

In order to get a sense about how its working, let's consider only one junction, in which the three portals are connected with each other. The junction has five game objects:

- Environment: contains the static 3d objects, assets.
- Cube: is used to know if you are in the actual junction or not. This is important because based on this information, the other cameras, which provide the image for the portals that are part of other junctions are not rendered. (If you have 16 junction, then 16 * 3 camera views should be rendered, which is excessive and cause performance issues with guarantee.) It can also be used to trigger game state changes, for example if the player has arrived into the last junction.
- GateA, GateB, GateC: The gates contain the actual portal planes and the camera associated with that portal. The role of the camera is to provide the view from the outside to that portal.

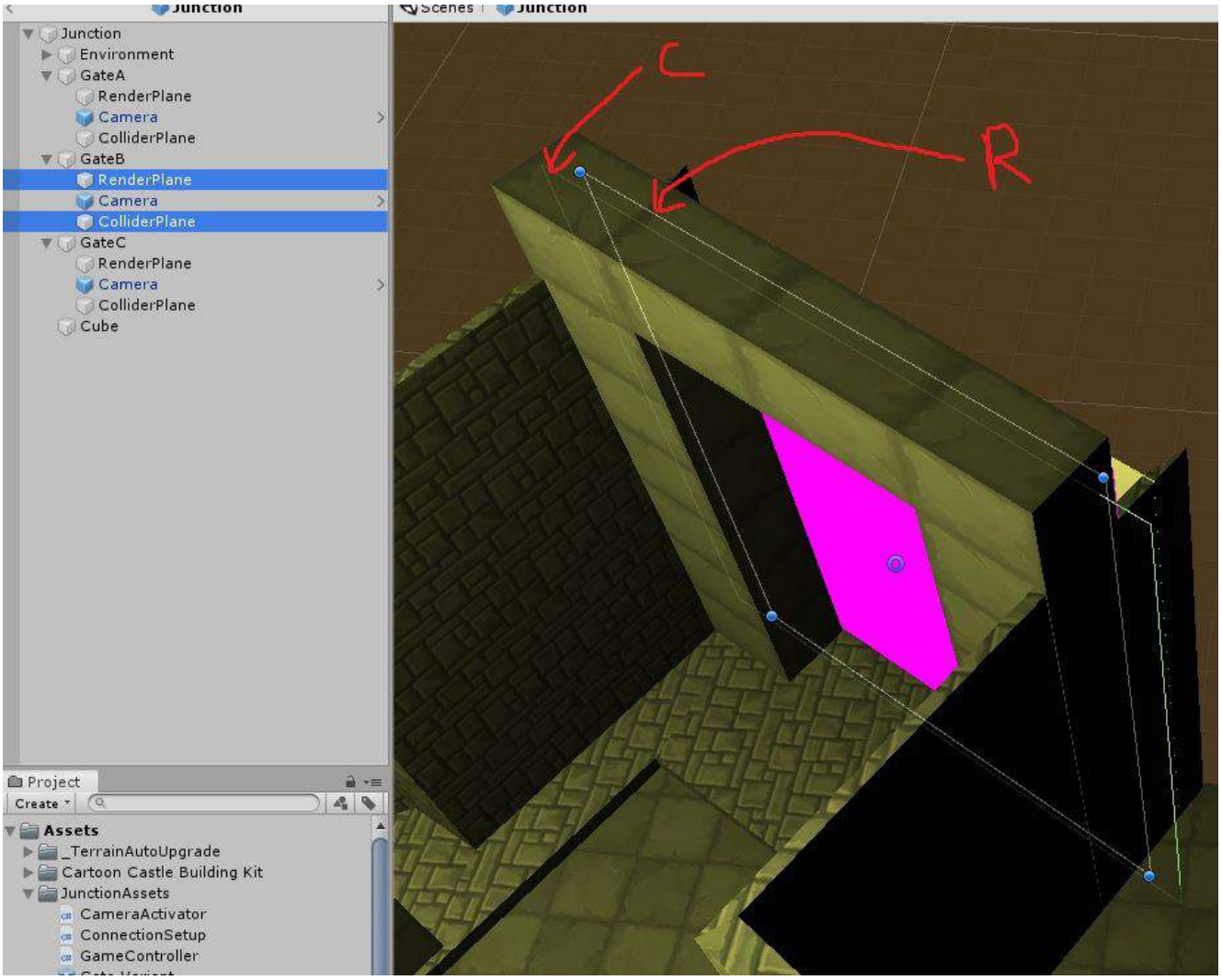


Fig 23. GateB from Junction prefab

On the **figure 23** 'C' stands for ColliderPlane and 'R' for RenderPlane. The ColliderPlane is responsible for teleporting the player from position x to position y (from portal x to portal y) with the right player-look-orientation, while the RenderPlane create the illusion of the portal. The implementation of ColliderPlane is much more simpler, since it involves less game elements than the other plane.

The script of the collider plane requires only four component:

- Reference to the homogeneous transformation matrix of the player
- Previous homogeneous transformation matrix of the player
- Parameters of the collider
- Transformation matrix of the receiver object

The magic is done by the following line of codes in PortalTeleporterScript.cs:

```
Vector3 vecToCurrentPosition = player.transform.position -
transform.position;
Vector3 vecToPreviousPosition = playerMoveScript.PreviousPosition -
transform.position;

// Rough distance thresholds we must be within to teleport
float sideDistance = Vector3.Dot(transform.right, vecToCurrentPosition);
float frontDistance = Vector3.Dot(transform.up, vecToCurrentPosition); //
UP because the plane has been rotated by 90 deg around the x axis.
float heightDistance = Vector3.Dot(transform.forward,
vecToCurrentPosition); //
float previousFrontDistance = Vector3.Dot(transform.up,
vecToPreviousPosition);

// Have we just crossed the portal threshold if it is true
if (frontDistance < 0.0f
    && previousFrontDistance >= 0.0f
    && Mathf.Abs(sideDistance) < /*approx portal_width*/ 7f
    && Mathf.Abs(heightDistance) < /*approx portal_height*/ 7f)
{
    // Teleport him!

    // Calculate the rotation
    Quaternion q = Quaternion.FromToRotation(transform.right,
reciever.right);
    // Calculate the new positon
    playerTransformRelativeToEntrancePortal = reciever.position + q *
(transform.position - player.transform.position);
    playerTransformRelativeToEntrancePortal.y =
player.transform.position.y;

    // Assigning the new position and orientation to the player
    player.transform.position = playerTransformRelativeToEntrancePortal;
    Vector3 newCameraDirection = q * player.transform.forward;
    newCameraDirection.y = -newCameraDirection.y;
    player.transform.rotation =
Quaternion.LookRotation(-newCameraDirection, Vector3.up);

    inPortal = false;
}
```

In the other hand, the visual part of the portal is much more complex, since it involves a custom shader, additional cameras in the scene around each the render plane, custom textures and materials and not talking about the scripts that are managing the creation and initialisation of these assets.
For example if we want to travel to point B which is in the middle of Gate B through portal A, which is in Gate A, then, we need to pass the ColliderPlane of Gate A. And along as the we are approaching to the Gate A, we can see the world through Gate B.

The relevant scripts are these:

- ConnectionSetup.cs
- CameraActivator.cs
- PortalCamera.cs
- RendererSetup.cs
- ScreenCutoutShader.shader

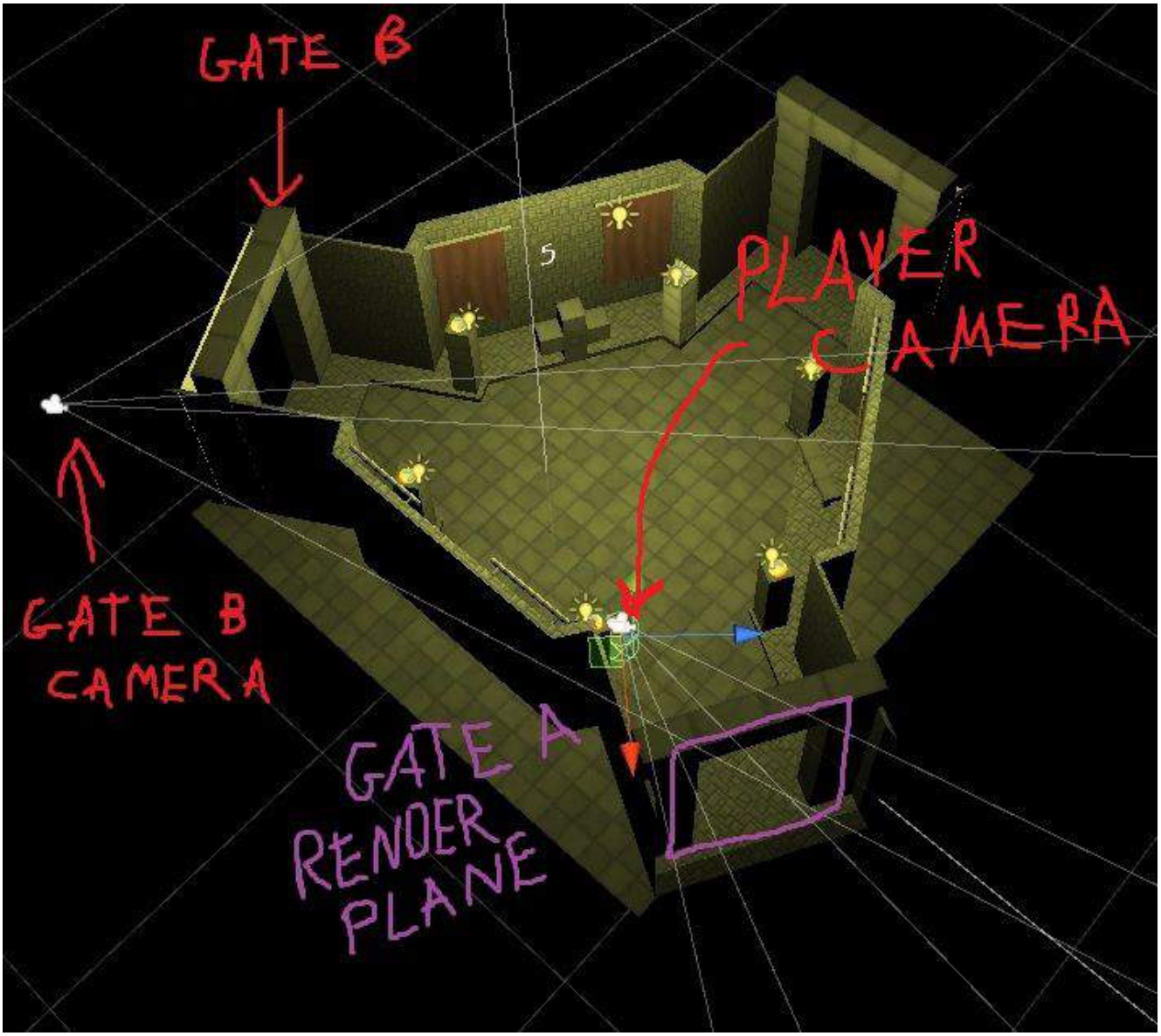


Fig 24. Example of portal from GateA to GateB

4.2.4 Game States and User Interface

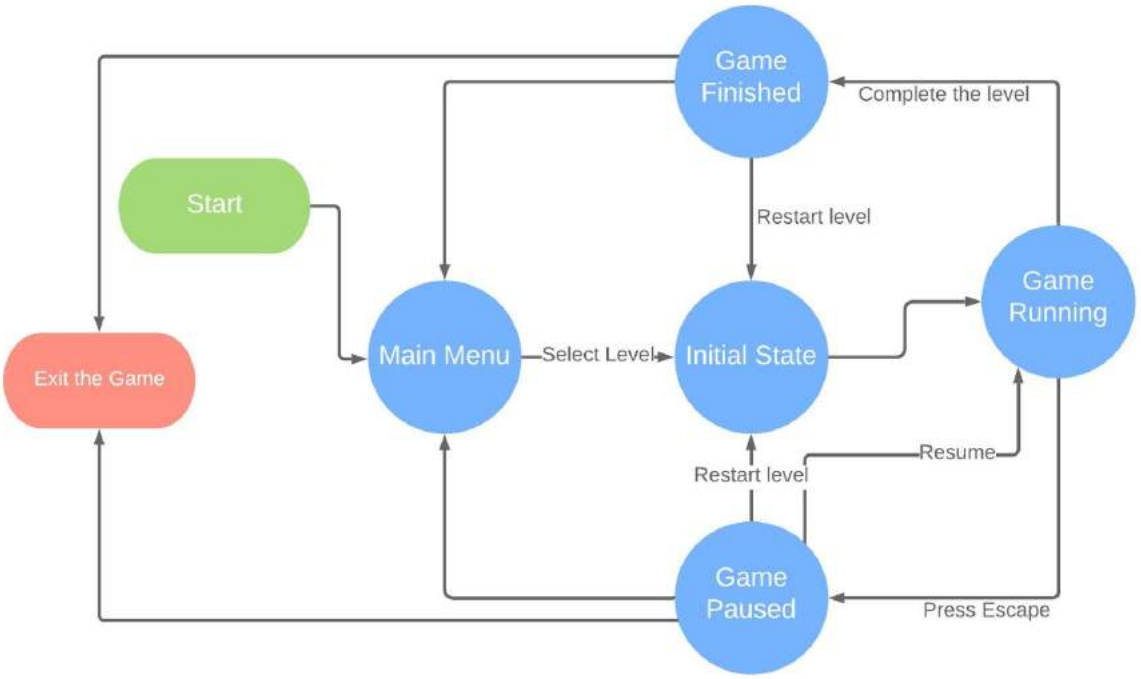


Fig 25. Game State Diagram

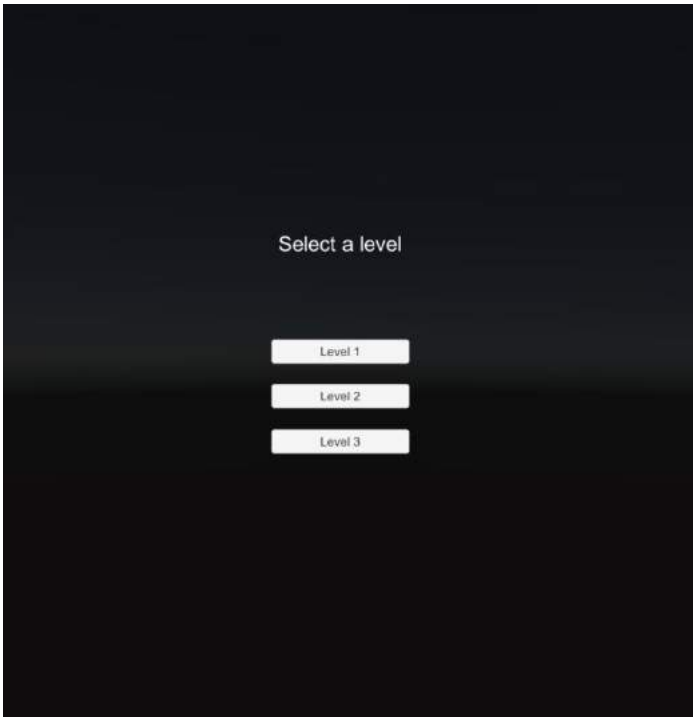


Fig 26. Main Menu - Level Selection

Fig 27. Game Paused

Fig 28. Level Finished

5. Game Evaluation

5.1. Analysation of the main tasks

5.1.1 Using the GUI

The GUI lets the player to navigate between game levels and to manage the state of the game. It is important for a UI to be self explanatory and easy (intuitive) to use.

1) Will the user understand what task he/she has to execute?
Since the game is not complicated and the UI elements contain texts, this should not be a difficult thing to understand.
2) Will the user be able to identify the correct interaction techniques (controls)?
The mouse cursor is always visible when one of the game menus are show. In addition to that, the buttons have a hover effect, which emphasizes the fact that they are clickable.
3) Will the user see the desired effect of his actions through an appropriate response?
Yes, the state of the game is changed instatntly after pressing a button.
4) Is there a risk that the user selects a different control technique than the correct one (the risk of error)?
There is no risk.
5) Will the user understand the feedback of the system for a correct interaction?
Yes, because this game is based on conventional game elements like: menu for level selection, pause functionalities, FPS view.

5.1.2 Character Navigation

The tasks that have to be evaluated in this scenario are:

- basic character movements (moving forwards, backwards, left and right sideways, sprint and player-look-orientation)
- Exploring the maze (testing the portals) by going from one junction to another

1) Will the user understand what task he/she has to execute?
In the game, the player has only one goal and that is to find the exit out of the maze. This gives an intuition to the player, that he has to move... Also, the movement is programed for the conventional “WASD” buttons, so the movement should be pretty familiar for the audience.
2) Will the user be able to identify the correct interaction techniques (controls)?
Yes. For the movement, the player should use the conventional FPS controls, which are the “WASD” buttons.
3) Will the user see the desired effect of his actions through an appropriate response?
Definitely yes.

4) Is there a risk that the user selects a different control technique than the correct one (the risk of error)?
There is no risk, but there is a chance that the player will not understand how the controls work if the candidate has no previous gaming experience.
5) Will the user understand the feedback of the system for a correct interaction?
After pressing the WASD buttons and maneuvering with the mouse, the player should see that the ingame character is moving.

5.2.3 Finding the End of the Maze

1) Will the user understand what task he/she has to execute?
In the game, the player has only one goal and that is to find the exit out of the maze.
2) Will the user be able to identify the correct interaction techniques (controls)?
The player must move in order to escape.
3) Will the user see the desired effect of his actions through an appropriate response?
Yes. A congratulation text will show up when a level is completed.
4) Is there a risk that the user selects a different control technique than the correct one (the risk of error)?
The only risk is that the player will never find the exit.
5) Will the user understand the feedback of the system for a correct interaction?
Yes. A congratulation text will show up when a level is completed.

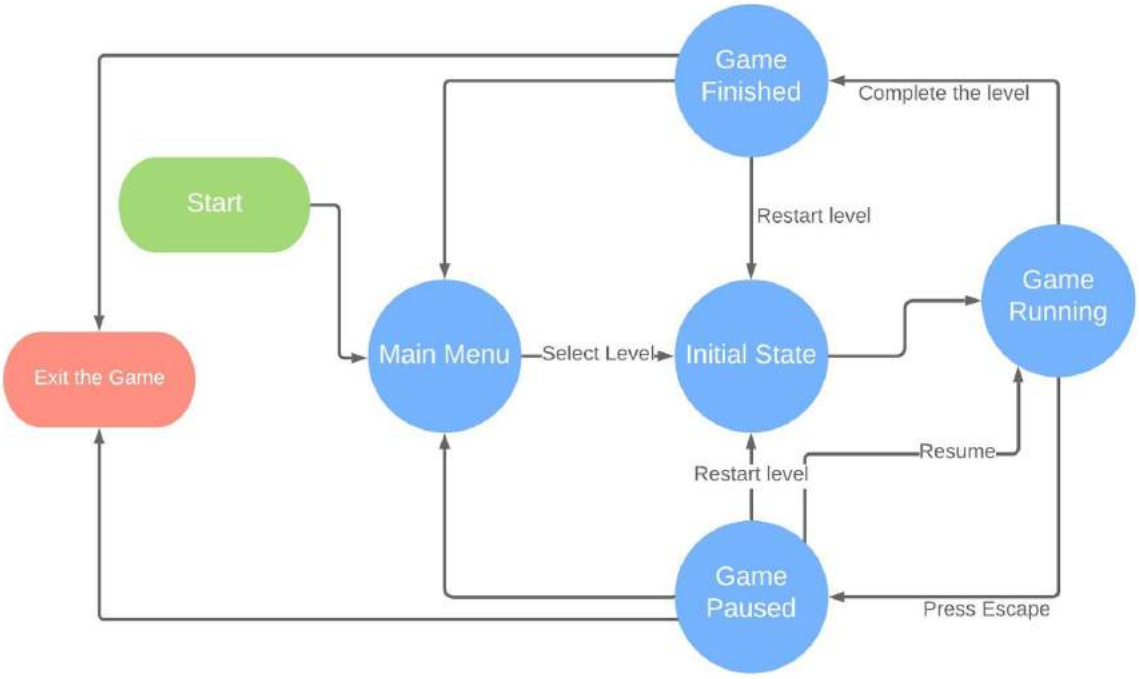
5.2. Heuristic Evaluation

5.2.1 Using the GUI

The evaluation in this section is about using the game menu:

- Main menu - where the player can chose a level
- Pause menu and
- when the level is completed

See the state diagram and the print-screens from the previous chapter.



Question		Ban Bogdan
1) Is the status of the system visible?		Yes, the application clearly shows what the GUI does with intuitive buttons.
2) Is the system (Game) created according to the real-world?		Yes, the menu uses familiar concept even for most casual gamers.
3) Can the user freely control the execution?		Yes, any button you press responds accordingly.
4) Are there consistency and standards?		Yes.
5) Are there techniques to prevent errors?		The only error you could make is to press the wrong button you desired.
6) Interaction based on recognition instead of memory?		Yes.
7) Flexibility and efficiency in use?		n/a
8) Simple and aesthetic		Yes.

design?		
9) Is the user helped to recognize, understand and solve the potential errors?		I don’t think it's the case for the menu.
10) Is there any help and documentation?		Not needed for using the menu.

5.2.2 Character Navigation

The tasks that have to be evaluated in this scenario are:

- basic character movements (moving forwards, backwards, left and right sideways, sprint and player-look-orientation)
- Exploring the maze (testing the portals) by going from one junction to another

Question		Ban Bogdan
1) Is the status of the system visible?		Yes, clearly.
2) Is the system (Game) created according to the real-world?		Yes, also, the room gives a vibe of haunted or abandoned, making a more thrilling atmosphere.
3) Can the user freely control the execution?		Yes. The movement feels nice to handle, it's not too fast not too slow and you can also run.
4) Are there consistency and standards?		Yes.
5) Are there techniques to prevent errors?		Making the game stable so the player cannot get stuck is a prevention. However, sometimes the player may

		enter a room but instead re-enters the same room from which he left.
6) Interaction based on recognition instead of memory?		The rooms are always familiar, but the puzzle is not.
7) Flexibility and efficiency in use?		Yes, the left shift for running.
8) Simple and aesthetic design?		The room as said gives a good vibe and the colours are pleasant to the eye.
9) Is the user helped to recognize, understand and solve the potential errors?		I did not find any error that needs feedback for it
10) Is there any help and documentation?		No, but the user should figure out how to escape the maze.

5.2.3 Finding the End of the Maze

The last task that has to be evaluated is about finding the end of the maze.

Question		Ban Bogdan
1) Is the status of the system visible?		No, and it should not, as it is a puzzle game. You do not know if you go to the right path.
2) Is the system (Game) created according to the real-world?		Not quite, the maze is infinite, but that makes it interesting, feeling like you may be trapped there forever.
3) Can the user freely control the execution?		Yes.
4) Are there consistency and standards?		Yes.
5) Are there techniques to prevent errors?		n/a
6) Interaction based on recognition instead of memory?		As said the rooms are always familiar, but the puzzle is not
7) Flexibility and efficiency in use?		Yes, the left shift for running.
8) Simple and aesthetic design?		Yes.
9) Is the user helped to recognize, understand and solve the potential errors?		No errors needing feedback were found.
10) Is there any help and documentation?		Not needed, only if the developer wants to help players who really want to solve the puzzle but cannot.

sometimes the player may enter a room but instead re-enters the same room from which he left.

6. Improving the game

6.1. Developer response from the heuristic evaluation:

Thank you for your feedback! I have changed the look of the junctions to have a much more abandoned feeling.

6.2. Game improvements from last time

The game was improved with various changes as it had some bugs:

- The timer is not counting anymore when the game is paused
- I put a plane behind each portal planes, because sometimes the player position is overwritten by an animation, which results in glitches. Fatal errors are prevented with that gate, so that the player cannot fall down between the junction into the void.
- The character cannot jump anymore, since the animation had a wrong behaviour when the player tried to jump through the portals.

7. Conclusion

We have successfully realyzed a proof of concept. Solutions of puzzles often require the recognition of patterns. People with a high level of inductive reasoning aptitude (measures how well a person can identify patterns) may be better at solving such puzzles than others. But puzzles based upon inquiry and discovery may be solved more easily by those with good deduction skills. Deductive reasoning improves with practice. The market of puzzle games is focused on people who like solving logistical or mathematical problems and since the game that I proposed is based on mathematical problems, the only people who will like this game are those that are familiar with high-school grade mathematics.

In Maze of Numbers, the player controls the unnamed protagonist form a first-person perspective. The player finds himself in the middle of a maze, in a junction of three corridor, from where he can start to walk in three different directions. The main game mechanics of this puzzle game is the ability to maneuver the protagonist around the spaces from one junction to another, where in this version of the game, numbers can be found making distinction between junctions.

After moving around a while, the player should notice that this game is different from the others regarding the typical notions of euclidean space, because the maze will present itself as a never ending bended space. The maze should brake down all the expectations after noticing that following only our intuition can easily result arriving to a point where we just started. The goal of the player is to find the exit.

7.1. Ways to enhance the game

I wasn’t able to implement many things in this game that I originally intended, here is a list of game elements that are missing or yearn for improvements:

- Graffities on the wall
- Individually customized junctions
- Different type of clues in each junction (involving a variety of new game mechanics)
- More precisely designed and implemented portals, because, currently the usage of portals are not completely fluent (the player can see a little jump when he/she enters the portal).

The source code and the complete project can be downloaded from my github repository: https://github.com/ErvinRacz/maze-of-numbers/ .