Alpha Release

Progress Report

Our initial goal for the alpha release was to transform our tech demo described in the interim report into a fully playable game. This was done by focusing on two main factors:

1. Since the initial tech demo did already implement all the key gameplay parts for our desired game, we wanted to delve into the creation and connection of animations, assets, shaders and other visual effects into the game.
2. Furthermore, we wanted to create an algorithm that could generate random labyrinths and would thus allow us to move away from statically created labyrinth levels. The labyrinth generation algorithm has to follow our initially defined constraints, in that it has to implement different looping sections inside the labyrinth that are connected by barriers that have to be opened by a key item placed in the corresponding section.

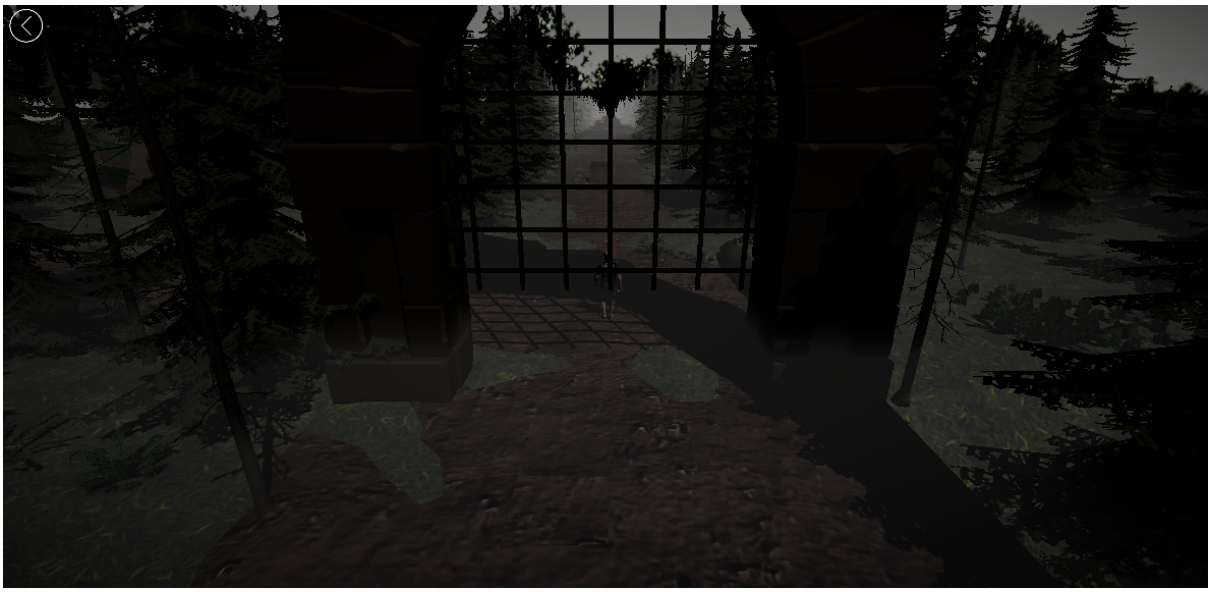
Assets

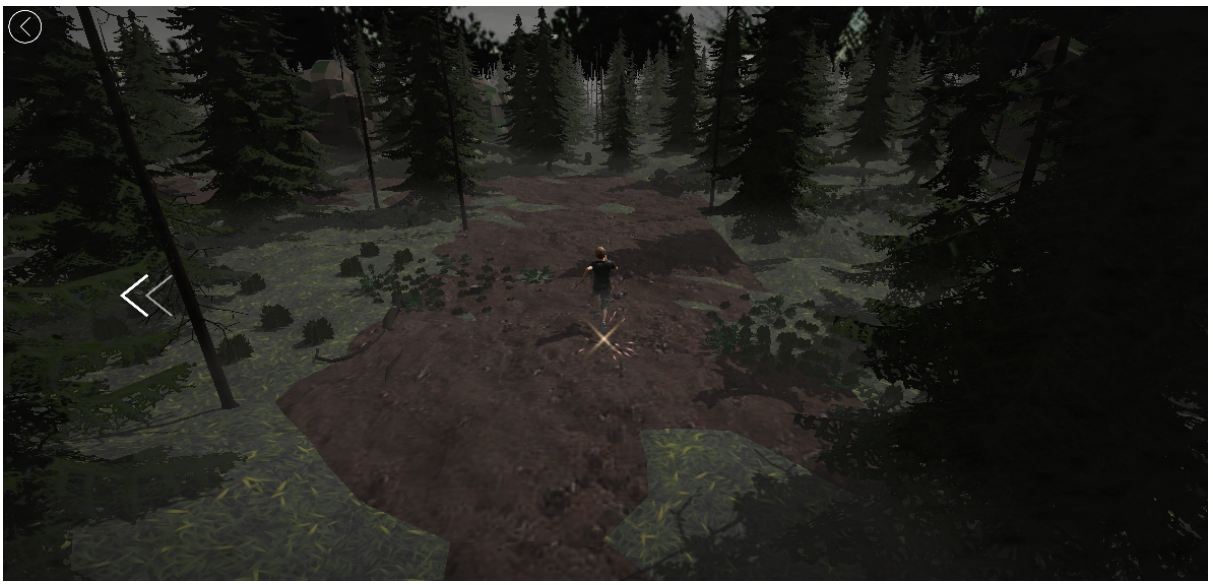
Animations

We used a free skinned character model from Adobe Mixamo. A young man looking like he spent some time jogging in a nice forest. We also found some suitable animations. These were assembled in the Unity animator, using blend trees and additive animation layers. Now we can trigger or adjust specific animations depending on speed and different actions taken by the player. They are mainly designed to give additional visual feedback and immersion. Some minor flaws still have to be fixed, for example the blending of the running speed.

Labyrinth Assets

We used a variety of free to use assets for our player model, labyrinth barricades, items and the tile design of the labyrinth. We decided to use a low poly style, since we could find the most free assets that would fit our artistic vision of a creepy horror dungeon runner.





Shaders
We used Unity's Shader Graph tool to construct an outline shader for the labyrinth obstacles in order to improve their visibility.

Visual Effect Graph
To further enhance visibility and player feedback, we implemented various particle effects with the help of Unity's Visual Effect Graph. These are used to highlight important events like turning or colliding with an obstacle and objects like items.

Music
We used free music from the Unity asset store as well as online databases. We layered different music and background sounds to achieve an ambience. Additionally, we implemented sound effects, e.g. player actions, the slender man or obstacles.

Labyrinth Generation

Generating the labyrinth was challenging. Our game design requires every section to lead to a loop without any dead ends, so players would always be allowed to turn around and backtrack. This leads to a large amount of interconnectivity. At the same time, the level as a whole should feel and function as a labyrinth. Additionally, there should be barricades that meaningfully restrict access to areas and items to unlock those barricades have to be placed in a way that they can be reached without having to pass the associated barricade.

Our first idea was to generate so-called “points of interest”. Those are random points in 2D space that will have an intersection and might hold an item. Then, we create the Delaunay triangulation of those points. This describes the general topology of the level and ensures that every edge is part of a loop. Lastly, to work with our tiles, we orthogonalize all edges and thus make it conform to a grid while keeping the general topology.

The downside of these levels is that they look rather bland and uninteresting. Furthermore, placing barricades and items in a challenging but solvable way would prove rather difficult. Because of that, we expanded upon our idea by dividing the level in sections. Each section is generated using the Delaunay triangulation and thus quite interconnected. However, passages between sections are rather sparse. This offers interesting locations to place barricades and makes the level feel a lot more “maze-like”.

So, the finished generation works in three steps. First, a high level description is generated, specifying general properties of the sections while being unconcerned with their exact layout. It describes their position in relation to each other and the position, all passages between them and the items placed inside the section. Passages can be blocked by a barricade. Passages and sections are created one by one. This makes it easy to place items in a solvable way. A new item can be placed in any section whose description has been finished previously.

Secondly, the algorithm above is used to create the actual layout of each section. This can be done entirely independently, section by section. The result is a 2D array of tile descriptions, specifying opening directions, items and barricades.

In a last step, the TileManager developed for the Interim Demo is used to instantiate all tiles and thus actually create the GameObjects making up the level.

Other

We enhanced our UI to better show the available options to the player in the case of a quick time event. This was done by implementing a separate pick up action for items, that would show the 2D sprite of the items in addition to the 3D model in world space, as well as connecting the 2D sprite of each item to its corresponding barricade.

<u>Encountered Challenges</u>

The development of a randomized labyrinth generation algorithm that fulfills our constraints proved to be harder to implement than initially expected. Although we did manage to develop

a sufficient algorithm, the effort needed to complete the task was greater than expected and did reduce our resources for remaining development tasks.

Another challenge was that the barricades and items we initially planned were too difficult to implement for now. First of all, it was not easy to find suitable assets. More importantly however, items like a whip used to swing over a chasm or a ladder used to climb over a rock need custom code and animations that were impossible to develop in time.

Since we did not work with a complete randomized level generation from the start, some already working algorithms (like the obstacle spawning) had to be adjusted to work on the new levels.

Although our current level generation does implement all of our required constraints, the resulting levels do tend to consist of multiple long roads that do not offer many locations for quick time events. Since the quick time events are the main interactive system of our game that is designed to hook the player and draw him into the gameplay loop, this design has to be revised. A possible solution that would not need to redesign the entire game would be to introduce another form of quick time events, for example with the help of dynamically spawned obstacles that the player has to react to. Also, mechanics of ducking and jumpring could make longer stretches of road more interesting. A last resort would be to decrease the size of the tiles or to adjust the procedural generation. However, this would take some more work as it would influence other systems in the game.

Future Plans

One of the key features we didn't implement yet is the spawning of dynamic obstacles that appear before the player, where the player has to react in time to move out of the way or perform a jumping or ducking action in order to evade.

Furthermore we plan on expanding the amount of supported barricades, obstacles and items that the player can encounter.

Lastly, general bug fixing and performance improvement has to be done. Until now, testing was almost exclusively done on PC, however, we are planning for the game to run on mobile devices primarily.