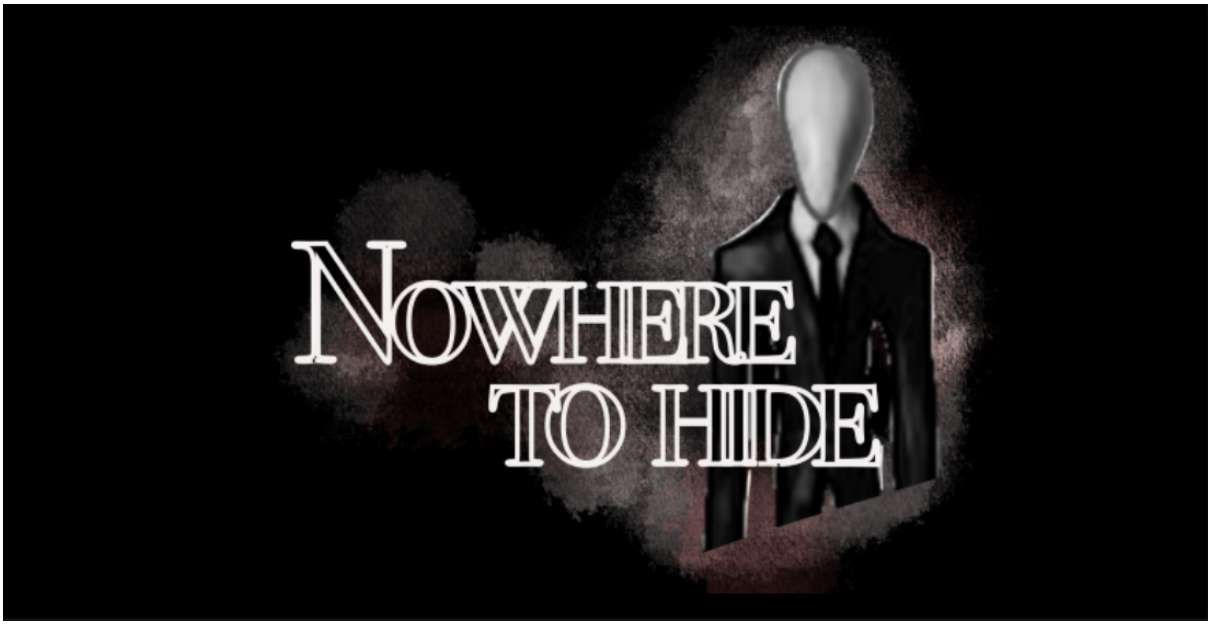
Interim Report

Progress Report

Our goal for the interim demo was to produce a first working version of our game that already implements our final gameplay loop but does not focus on technical achievements or graphical effects. In order to reach this goal, we first wanted to implement a working UI System, a way to handle user input, a way to generate levels and a win and a lose state. We investigated different algorithms that could be used for the procedural level generation but did not implement them yet. Furthermore, we researched different assets that could be used in the final game and decided to use a low poly look.

UI System

In order to load into the game, we designed a main- and loading menu. Furthermore, we implement an ingame menu that lets the player quit the game at any moment. Next improvement will be to just pause the game and offer the option to return to the main menu or to quit the game.



User Input

In order to quickly switch between a cellphone and a PC user input system for development and testing purposes, we used the “new unity input system”. This system defines input actions that can be invoked from different input devices. The main gameplay in our project consists of evading obstacles and making quick time decisions. The player is automatically moved forward through the labyrinth.

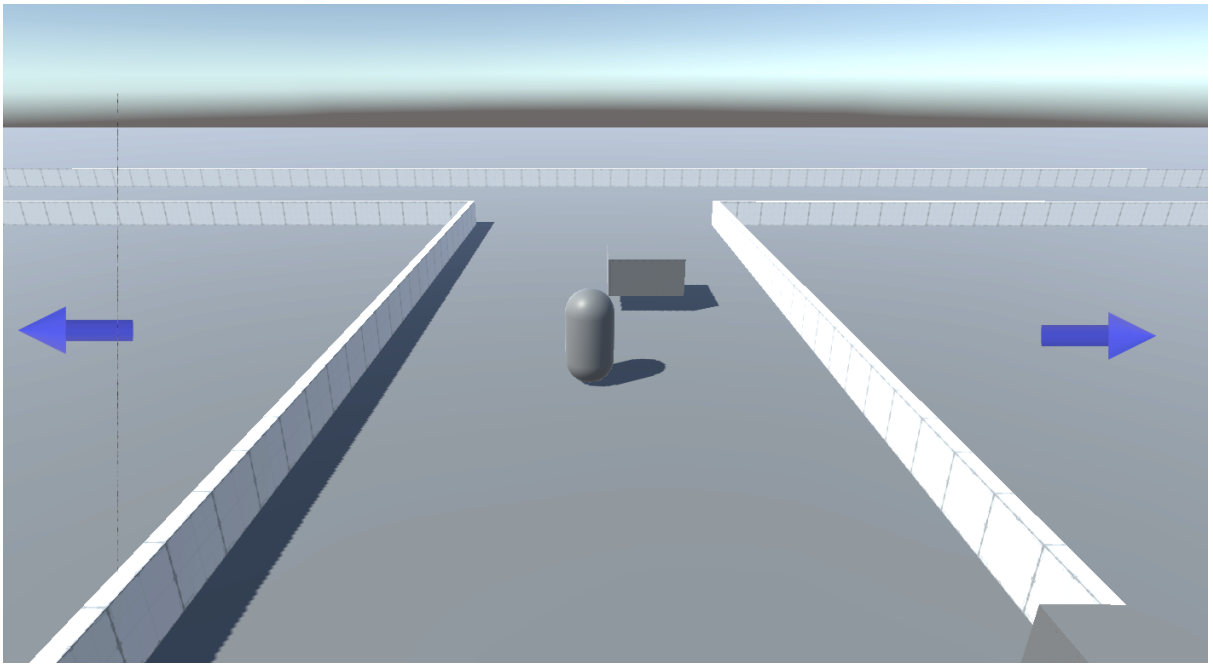
For the cellphone version of our game the player can dodge obstacles by using the gyroscope, while using swipes on the touchscreen to perform quick time events. For the PC version the player can use the mouse and keyboard to achieve the same input. If the player fails to perform a quick time event in time, for example by not

deciding on which way to go on a crossroad, the game is randomly selecting one of the available options and is punishing the player.

Quick time events
Available quick time events are defined by the tile the player is currently traversing. For example a crossroad may offer the options to turn east, west and south. However, the actual behaviour when making an input is implemented in the player script. To achieve this design, we decided to use the visitor pattern.

The player retrieves available actions from the current tile and maps them to the swipe-inputs left, up and right, depending on the direction the player is facing. When one of the inputs is made, the action gets delegated back to the specific player behaviour and the available actions are cleared. Similarly, available actions can map to UI elements to be displayed on screen.

This system gives us lots of flexibility for implementing possible future quick time events independently from the ones that already exist.



Level Generation
Since we could not decide on an algorithm to generate the procedural levels yet, we decided to lay the groundwork for it by implementing a system to manage different tiles with various attributes that can be arranged to build our level. For each tile, information about the path, size, available quick time events and visual style is stored. These tile descriptions can be arranged in a two dimensional array, giving a world-space description of the level. For now, we created this array by hand, later, it will be generated procedurally. Next, the Tile Manager is used to turn the array into the actual level. Each tile is retrieved as a prefab, rotated appropriately and placed in the level.

At runtime we spawn random obstacles that the player has to evade on each tile. If the player does not evade the obstacles, he is slowed down and the Slender Man comes closer.

Win and Lose

In order to finish the gameplay loop of our game, we implemented the Slender Man character. The Slender Man has to chase the player through the labyrinth. The distance of the player to the Slender Man is visualized with a random noise effect like static on an old TV. The player slows down and loses speed if the player does not evade obstacles or makes bad decisions during the quick time events, which makes it easier for the Slender Man to catch up to the player character. The player loses the game if the Slender Man can reach the player.

Encountered Challenges

Initially, we wanted to present an algorithm to procedurally generate the labyrinth levels. After initial research however, we came up with multiple different ways to tackle the problem of random level generation. We knew that getting the procedural generation to work would require some trial and error and lots of tinkering. Thus, to ensure we'd be able to successfully lay the groundwork for our game, we delayed implementing the procedural generation to a later time and focused our resources on the basic game loop described above.

We researched different assets and graphic styles to use with our game and revised the graphic style of our game to be low poly oriented, for which we found a suitable asset-base and, if needed, could also create assets on our own. However, we did not yet put the assets into the game and did not focus on any visual effects. As a result, while the basic systems are in place the game is not yet very engaging to play.

We also revised the software structure of our project by splitting the game code into different assemblies to improve compilation speed and clearly show dependencies between parts of code.