Project Summary

1. Introduction
   For this project we have chosen to make an open source game with the Godot engine v3.0.6[1]. For this, we used Godot's GDScript Language which is very similar to Python. We implemented a labyrinth similar game[2] using a 9x9 grid. In the game one player is playing against an AI controlled player. The goal is to reach the goal faster than the AI controlled player. To make the game dynamic, both the player and the AI have to chance to shift the tile's before they start playing. With this possibility you can try to block the other player and reach the goal faster.
2. Body
   a. The Open Source Engine
      We decided to use the Godot open source engine. It has many advantages for our desired project: The engine itself is licensed under the permissive MIT license, which allows us to distribute our project without any concerns. The developers of the engine also allow any projects created in the engine to use any license. It also provides a basic 2D (and 3D) game engine, which allows us to skip fundamental game development parts. The engine itself uses a three structure with different scenes.
   b. Folder Structure
      We used many different folders to be used for the game. We kept fonts that we used in one folder (in `/fonts`) and the images used in another folder (in `/images`). Then we have a folder for the scenes (`/scenes`) which is used to handle the different scenes we to create the project. In the scenes, we place different objects and sprites. We can attach code to each object, which shall be executed. This code is stored in `/scripts`. In the script folder we have sub folders like `/scripts/player` which contains scripts only for the player.
   c. Creating the Grid
      The grid is created by populating a variable called gridSet with vectors that represent each cell of the playfield. At the beginning of the game walls are randomly generated using a built in function of Godot. A separate script draws lines on the grid for a better look and an easier way to tell which cell the player and AI are located.
   d. Player movement
      The script which handles the player movement, reads input from player. Based on the player input it move the player to the next vector 2 in the direction that the player pressed. It has checks to prevent the player from moving in two directions at once, and stops the player from trying to move diagonally.

[1] https://godotengine.org/download/linux, accessed 05/11/19
[2] https://en.wikipedia.org/wiki/Labyrinth_(board_game), accessed 05/12/19

e. Implementing the HUD/Title Screen/Images
The HUD is completely outsourced into the HUD Scene. With this implementation, we had the chance to build the HUD and the Grid separately. So we could avoid git-conflicts in the merging process effectively. In this scene we have the buttons to shift the tiles into the proper direction. In the main script, we simply emit a signal for the appropriative shift movement. This signal is called in the TileMap script, which maintains the grid. Here we have two helper functions, one for the vertical and one for the horizontal movement. These get as an input the row/column, which shall be moved. With this implementation, we can reduce the lines of code and also have a proper code modularization.

f. Implementing the AI Movement
The way the AI moves is very similar to the way the player moves. However instead of button presses, the AI uses a GDscripts AStar class. AStar is a computer algorithm used for pathfinding. When it is time for the AI to move, it calls a function from the TileMap class which creates a AStar of the entire grid. Each tile is represented as a point, and adjacent tiles are connected via an edge. Each point has a weight attached to it, if the tile is a wall its weight is 999. If the point is not a wall its weight is 1. The AI calls an AStar function that creates a path from its current position to the goal. If there is no path available then the AI will move to the closest position to the goal possible.
We wanted to have the AI intelligently shift the grid on its turn by checking if it had a path, to the goal, then either shifting a the grid near the player or itself depending on which was more beneficial to the AI. Unfortunately we ran out of time and were unable to implement this.

g. Art Design and Main Menu
All art design is completed by the open source art software Piskel, including player, ai, treasure box and start button. Main menu is created by open open source application inkscape. Main menu includes the brief instruction about how the game will be played and a start button beneath the Instruction. Therefore, players can have a rough idea of the game.

3. Conclusion
In this course, we had the availability to explore a new engine for developing games with a known language as its scripting language. Because it was the first time using this program, it was sometimes difficult to understand how to use the programs built in functions e.g. handling signals. Also the connection to the grid was new to us at begin. But we could handle this problems and then developed the whole logic and movement implemented in the game on our own without any additional libraries. Also we learned, how to use git properly during the project but had also the experience, what happens, when you don’t pay attention the whole time and you have to solve conflicts by hand (which can be painful). To sum it up, we implemented successfully the game mentioned in the proposal.