**AP Project 2023-2024: Pacman game**

Thomas Urkens 20212261

**Introduction**

The project we had to make this year is the famous game Pacman. In this report I will go over all the design decisions and implementation problems I faced while working on this project.

**The setup**

When starting the project I didn’t know how to begin since there were a lot of things to take into account such as the class hierarchy and design patterns. I started with adding simple things like initializing some necessary classes like the random class and score class with the scoreboard before moving onto the main game and setting up its class hierarchy.

**SFML**

Since I had never used the SFML graphics library before, I had to experiment with its basics first. Fortunately everything became very clear very quickly because it’s quite easy to use. There was only one problem I faced using SFML which is that sprites don’t keep their texture when using functions to initialize them. This was fixed by making a struct with the sprite, the texture and the image. The same goes for texts and their font.

**Creating the game**

I started in the game class setting up the main game loop and check for input keys. After that I made the statemanager and the necessary states. I created all the visuals for the states except the level state and linked the statemanager with the game class so it switches states depending on user input in the main game loop. Since one button press generated multiple iterations of that key pressed in the game loop, I used some variables to create cooldowns for certain key presses. Timing these cooldowns took a while, mainly because not all states run at the same speed.

**Game logic**

I started the game logic by making the world and its entities and placed them in the world. Placing these entities was a lot more work than I first thought because there were more entities than I expected and I had to specify all coordinates. I organized the entity placement code a bit for a better overview. I also made a boolean to disable coins for testing. Since the world was bounded by [-1,1] and the movement of Pacman is dynamic, I was trying many different hitbox sizes for all entities to get Pacman’s movement and collision right but it never worked completely. I needed extra helper functions for this. While Pacman is in a straight corridor it is only supposed to move in the direction of the corridor so when the user wants to change the direction of Pacman it first checks if Pacman would be able to move 5 game ticks in that direction to see if it is a viable direction.

This way Pacman can only move straight through corridors while these corridors are still wide enough for Pacman to enter them from intersections. The game class also stores 2 movements so when the user wants to change direction and it fails, it will keep trying that direction first before moving Pacman in the current direction until it works or Pacman collides with a wall.

For the ghost AI I needed to create a different collisioncheck and move function since they are a bit different from Pacman. The ghosts also have some extra variables like a second smaller hitbox used to detect a collision with Pacman, a chasemode so they leave the center at the correct time and a lastswitched so they don’t switch directions in large spaces (like the spawn area) and get stuck. When eating a ghost in fear mode it becomes normal and teleports to the ghost spawn, but Pacman sometimes still died to the ghost because it switched to fear mode too quickly so I had to use another variable lastdead to make sure that this didn’t happen. Just like with Pacman the ghosts will check if they can move 3 ticks forward to the direction they want to switch to in order to move properly through the corridors. Using these ghost variables and helper functions the ghosts will move like the assignment described.

There is also a singleton Levelstats class which keeps the data like score, lives and difficulty of a current playthrough. The world will update the Levelstats correctly. Every completed level the difficulty will increase by one which means the ghost’s speed will increase by 5% compared to the previous level and the fear mode duration will be 20% shorter.

**Game representation**

The main part of the game representation is the camera class. After some sketching on paper I made some formulas to display all entities from the world on the SFML window. The entities contain values updated by the world to let the camera know which sprite to use. In the beginning I cut the sprites from the png every frame but that made the program extremely slow so I initialized these sprites in some vectors in the camera constructor instead. I used some modulo calculations to use the correct sprites for the direction and movement animations from these vectors. I also used exceptions to check if the used files exist like the Arial.ttf font file which SFML needs in order to display texts or the Sprites.png to display the sprites.

**Root directory**

In the root directory you can find main.cpp to start the game as well as the source code divided in Game representation and Game logic. There is also a CMakeLists.txt to include the source files and to find the SFML graphics library. The root directory also has a clang format file by which all the code is formatted to and the CI platform.

**Conclusion**

I really enjoyed working on this project because all progress is very rewarding since you can see it visually change in the game. I also learned a lot by making this project like advanced C++ functionalities and using a graphics library. It’s also nice to let other people play a game I made.