Overview

WASD-movement
- uses its own movement system and velocity

Combat between enemies and players
- instead of turn based we came to conclusion that real time combat would be better for our game so we used it instead

Player Character
- Player character has HP, XP and its own inventory for consumables.
- It also has armor and weapon slots. Armor and weapons get better when leveling up
- Player character has collision with walls.

World interaction
- Picking up items (potions)

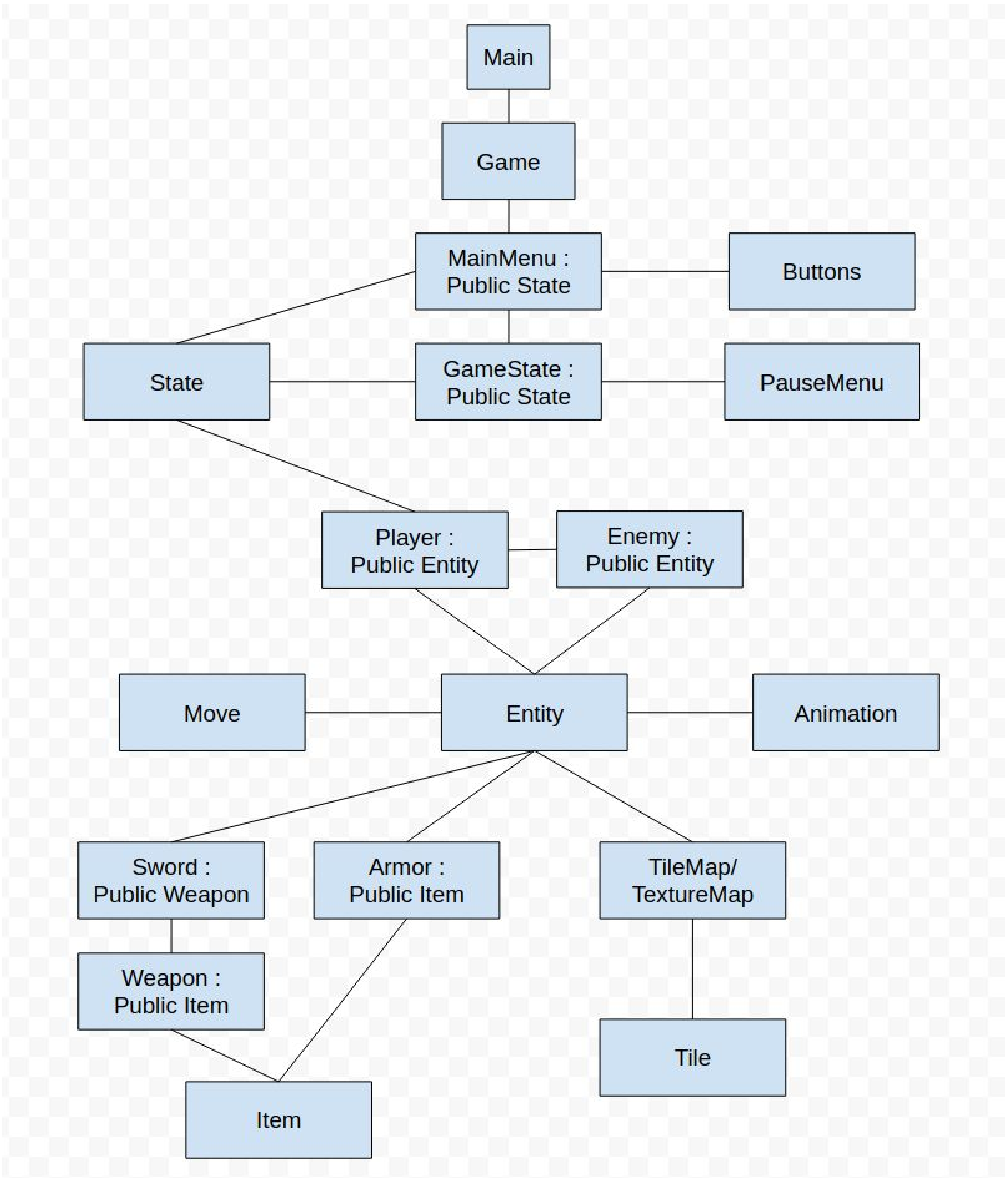
Enemies
- We overhauled the static enemy plan when we redesigned our combat system.
- We used the same movement system for enemies as we used for player.
- Enemies have own aggro range and start moving towards player when getting close enough
- Enemies have HP, armor, weapons and they give xp when defeated
- due lack of time we were not able to implement npc characters, which we had planned to make in our initial project plan, for the same reason we don't have other storyline than "defeat all the enemies".
- Enemies are ghosts so they don't have collision. This is purely a feature, we definitely did not run into problems with pathing, which would have required complex algorithms to solve nor did we notice any bugs with enemies getting knocked inside the walls.

Sounds
- Player has interaction sounds for hitting, getting hit and drinking potions
- we have music for main menu and the game state itself
- NOTE: if any other music is playing at the same time (outside the program) it causes errors and is not able to start the game. I believe this is caused by something in sfml library.

Game ends after defeating all the enemies on the map (including the final boss) or when the player dies.

States
- Main menu
  - Starts the game, plays neat music
- Game State
  - Runs the game
- Pause menu
  - Pauses the game

High level structure of the software:



Description of software logic

main starts the game, game creates mainmenu state and adds it to the stack. mainmenu has the ability to create new game states and add them on top of the stack. gamestate initialises the map, player and enemies and is responsible for updating and rendering them.
Tilemap creates map from map.txt file and is called from gamestate class. Almost every class utilises parts of SFML library.

Testing:

test were done by simply running the game and trying out different kinds of scenarios

Work log:

Joonas:

23.7: created game class and start of the state structure
24.7: improved state structure and created entity class. added gamestate. set up cmake and compile.sh
27.7: created mainmenu state, improved game structure
28.7: created player class, improved states. added fonts
31.7: added playermodels, overhauled state system stack, fixed need for resource copying. Created tilemap system, tilemap loads from text file.
1.8: Improved map, created texture map system which allows overlapping textures, thus a lot cooler map
5.8: Improved map, overhauled texture map system so it can read values above 9 (text based, char 9 turned into number 9 by subtracting ASCII 0)
added more textures and main menu background, added music.
made so animations match movement, added extra components.
6.8: path fixes, global variables, memory leak fixes. walking fixed, started collision testing, cleaned up code
10.8: moved stuff from player to entity class, player inherits from entity
View follows player, added deferred rendering, structure fixes.
fixed small animation bug.
11.8: more global variables, improved map
14.8: fixed issue with flickering, window no more updates its minimized/running on background
19.8: fixed broken code :)
21.8: memory leak fixes, fixed animations.
23.8: added inventory system, items, inventory, start of GUI
24.8: leveling up system, improved GUI
25.8: small fixes

26.8: fixes to animations, added winning condition
27.8: added sound effects, removed non-functioning enemy collision, improved animations
28.8: armor no longer bug when leveling up, more sounds, menu now looks good fixed includes and cleaned up code. Finished up the code.
Total time used: 85+ hours (estimated)

Tommi:

27.7: added main menu music
31.7-21.8: created pause menu class

Samu:

20.7.-2.8.

Had a lot of problems with syncing git to visual studio code and getting sfml to work. Eventually had to download visual studio and set up sfml that way, sfml did not work with git sync however, eventually fixed the problem

3.8: Added button class, way to create buttons easily. Edited main menu to use clickable buttons, added
statestack so clicking "play game" pushes the gamestate on top of the menu state

6.8: Collision box for player, creates "true bounds" for player, so that colliding works correctly. Created tle class to check if tile is wall or not and edited tilemap to to create the tiles and check the values.

13.8: Added collision system checking if hitbox collides with map bound or with walls using tiles. Checks which way the collision is and pushes the player back. Fixed tile class so that collision can check intersect and added capability to check tile at some index. This was done by using map of tiles and corresponding indexing. Created a few helper functions to get the tile index from pixel coordinates.

14.8: created movement component, a way to move player more efficiently and realistically using acceleration and deceleration. Edited collision to use movement component debugged bugs related to player collision (getting stuck on some walls, being able to walk through some etc.)

25.8: HP bar to show player hp graphicly and bar moving with the player, was relatively easy.

Jaakko:

20.7 - 26.7:

We had our first live meeting and we tried to get VSCode, git and SFML to work on every group member's computer. We figured out that it is easiest that everyone use Ubuntu or WSL.

27.7 - 2.8:

Some of us still had some troubles getting the software environment to work.

3.8 - 9.8:

With the help of youtube tutorials animation class was relevantly easy to code but we struggled a lot with syncing the player movement and animation. We ended up with a pretty complex system that controls the player animation.

10.8 - 16.8

I started to make a sword class but for that I had to create an item and a weapon class which the sword class could inherit so we could have multiple weapons and items.

We figured out that the best way to implement the attack animation was that when you attack you can't move the player at all and when the attack animation is finished you can move the player again.

17.8 - 23.8

Then I had to make some enemies which the player could attack. Enemy class was the same as the player class. The different part was the movement. This time I made the enemy to simply move towards the player if it was in a certain range. Then we had to figure out how the attack works. We needed a function that determines if the sword hits an enemy or not. The range of the sword is pretty much a half circle so the Pythagorean equation worked fine for that.

Armor class wasn't hard to implement based on sword class.