Orcs n Towers

Scope

Orcs n Towers is a tower defence game set in a fantasy setting, where orcs and other such monsters try to reach and destroy the player's castle, the player must defend against the monsters by placing different towers with specific roles. The player will have a set number of hitpoints that are depleted when enemies reach the castle. When all hitpoints are lost the player loses. This is one implementation, depending on the difficulty of the game the loss condition could also be when an enemy reaches the castle. In order to build towers and defend the castle the player must manage the in-game currency to buy and place towers (towers that give in-game currency may potentially be implemented). The player wins once he has survived all the rounds, the initial plan is to have 5 rounds.

A leaderboard system will also be implemented. Metrics for player success will be calculated using the time spent to beat a level and how many hitpoints the player has left. Each wave gets progressively harder, and the levels will have milestone waves that introduce a new enemy type, which will increase the level of difficulty. Certain enemies may be immune to certain types of tower/damage, some may be very quick while others slow. Different enemies will also have different HP and will do varying damage to the castle if they are to reach it, assuming this feature is implemented.

The player will have access to a variety of different towers with unique abilities and quirks. Examples of such towers are a slowing tower, poison tower and explosive tower to name a few. The slowing and poison towers will instate a status effect onto enemies that are hit, causing subsequent effects such as slowing the movement speed of enemies and/or applying poison which depletes enemy health over time.

The game map path that the enemies traverse will be randomly generated according to some strict criterias that ensure that the path is in a range of specified lengths and connects the beginning and castle. All towers can be placed anywhere outside of the path. Towers will also have unique ranges and fire rates giving them strengths and weaknesses. If time permits a tower selling feature may be implemented, allowing players to sell a tower and receive some percentage of the total value of the tower in currency. The game will also implement a pause and play feature.

Basic class structure

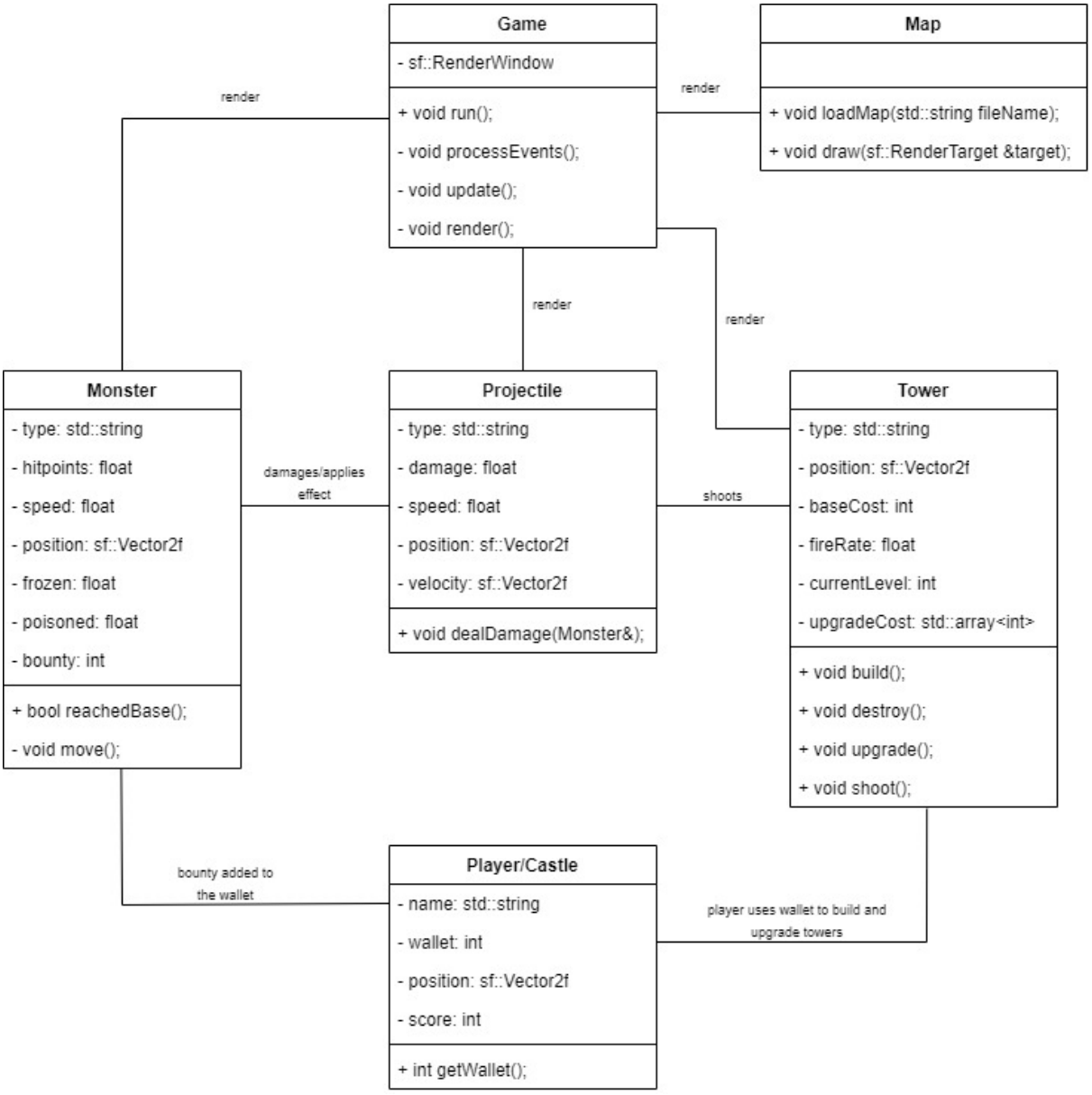

Figure 1: Planned class relationships with currently identified member data fields and functions

Tower class

Base class for different types of towers.

- Private members:
    - Size
    - Damage
    - Location

        - Rate of fire
        - Weakness = what enemies can this type of tower hit?
        - Pointer to projectile type, which is fired by the tower
- Public:
    - Upgrade method
    - Destroy method
    - Attack: Create projectile object corresponding to the type fired by tower, add the bullet onto a collection holding all currently existing projectiles in game

Projectile

Base class for different types of projectiles, fired by towers

- Private:
    - Position
    - Speed
    - Velocity (needed to draw correctly directed sprite)
    - Owner (which tower they can be shot from)
    - Type
- Public:
    - Getters

Enemy

Base class for different types of enemies. (Do the child classes for this change any other properties than the constructor?)

- Public:
    - Getters
    - Change state
    - Move
- Private:
    - Type
    - Speed
    - State
    - Hitpoints
- Update function: advances state

Game class

A game class handles updating the state of the game and rendering the state onto the screen. This class holds the SFML window object, and collections storing the different entities in the game. The entities can be towers, projectiles or enemies, which all have their own separate base classes from which different types are derived. These classes contain a

pointer to a texture associated with them, stored in a separate texture class for ensuring that they are loaded and stored only once at the start of the game.
Members:

- Run: Wrapper function, which is called from main
- Handle events
    - Mouse events: pause button, possible purchases of towers and upgrades etc
- Update state: iterate over collections of towers, projectiles and enemies
    - Update their state by calling class member functions
    - Handle destroyed enemies and projectiles: collections holding these need to be updated accordingly.
- Render onto window:
    - For each element in collections storing entities call draw function
- Collections of entities: as linked list? Addition of new elements should only happen at the end of the collection, but removals may happen in the middle
- SFML window object

Texture holder class

Loads required textures for drawing and stores a single instance of each for fast access. Probably implemented as a template class, so functionality can be reused for loading music / sound effects.
Members:

- Map containing pairs of texture type enum and texture pointer wrapped in unique_ptr

Player class

Store health, money and score. Health and money should be accessible by the enemy class.

- Private members:
    - Health
    - Money
- Public:
    - Getters
    - addMoney
    - Remove money/health

Use of external libraries

Based on our current understanding SFML should be sufficient to provide all the required functionalities and thus it will be the only external library used in the project. Mandatory SFML modules in the context of the project are Graphics, System, and Window modules. Additionally, an Audio module will be needed if the final version of the game features sounds.

Division of work

Pavel: Tower class & Main game class
Leo: Enemy class
Ellen: Projectile class & player/castle
Tuan: Map class
Otto: Music and graphics

Schedule/milestones

- By 5th of November:
    - Learn about used libraries
    - Get comfortable with project plan
    - Create file structure
        - required hpp/cpp files
        - link together with include
        - define classes
- By 10th of November:
    - Outline of parent (tower / enemy / projectile) and main game classes' functions & members
    - Basics of GUI (hardcoded)
- By 15th of November:
    - Create child classes for tower / enemy / projectile
    - Economy & health functionality
    - Further graphics
    - Implement start / stop game in main game class
- By 22nd of November:
    - Connecting towers and projectiles so that towers can shoot (not necessary towards enemies)
    - Connect enemies with path (enemies move along path)

- By 26th of November:
    - Connecting towers, projectiles and enemies (towers can shoot at enemies)
    - Enemies should lose HP when hit → eventually die
- By 28th of November:
    - Be able to place / remove / move towers
    - Connect with economy to pay for towers
- By 5th of December:
    - Game class improvements and additions
    - Different levels (variance in amount/type of enemies)
    - Upgradable towers
    - Load maps from file
- Testing and finalising until due date