Universiteit Antwerpen

Advanced programming
2020-2021

Wils Nick
20162945
2e BA INF

Report Turbohiker

Initializing

The game starts from the game class. Here I will initialize the graphical parts and steer the game. The constructor will initialize the world and window. It will also set the track length. The length and width of the singleton class Transformation will be set as well. Because the playable field is smaller than the window, I chose to give the Transformation class the width of the track. This is shown in “Figure 1”.

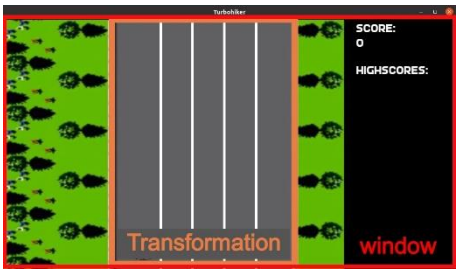

Figure 1 screen-size vs Transformation-size

“main.cpp” will Initialize a game and then call the function run. This function will give back a Boolean. As long as run gives back true a new game will start.

To initialization of the game happens in multiple parts:

I will load the background image only ones and make 3 sprites from it. These sprites have the same x position, but the y position is different. They are set above one another. At the start, the middle sprite is fully in view. When I go up the next spirit will come partially in the view. When the player is in a quarter of the middle sprite all sprites jump up by the screen height. This makes that we are now in the lowest sprite.

I also initialize all the factories for: the player, enemies, speech bubbles and scores and obstacles, end-line, lane-line. This is done with the respected SFML factory. I have 2 Factory interfaces: one that creates hikers, one that entities. There are 2 because I wanted to get back an instance of the Hiker class, so that the world can use Hiker-functions on player for example. I gave all the SFML parts, like the window and sprite, too the constructor of the factory and set them in the member variables.

Next, I call the “initGame” and “generateObstacle” of the world class and give it those factories. Now I can generate entities without knowing anything about the SFML parts. The number of lanes is randomly set between 4 and 7. The player position is also set randomly in one of those lanes. The number of obstacles is 25 times the number of lanes. I generate those obstacles in a while-loop, where I randomly choose which type of obstacle is made. All those obstacles are set from just outside the start-view to the finish. If an obstacle collides with another, it is removed and a new one is generated. If there are too many collisions it means that there is probably to little space to make new obstacles and I will stop making them.

When initializing an entity, a new subject and observer is made and linked with that entity.

I also read all the high scores from the text-file.

All textures will be kept in a list in the Game class, only ones.

Singletons

For the singletons I used a shared pointer as an instance. For the random class I used the random library to generate random numbers.

Wils Nick
20162945
2e BA INF

Game loop

After the initialization the game loop starts. There are a couple of parts that need to happen here.

Clock

Every tick I calculate the time between now and the previous tick. This delta Time will be scaled to a more usable number. Together with the time right now I will update the time and delta Time via the world in all the entities.

Calling actions on world

To update the world, I will call different functions on the world like the following:

- Remove locks
- setTimers
- Speedup
- Shout
- Update
- Remove entity

To keep entities from switching lanes to quickly, yelling too much and for the buffs, I add some amount to the current time (lock them until that time).

In those functions I also give the world input from the player. Instead of using multiple Booleans I chose to use integers. For example, for left and right input I have 1 integer that is -1,0 or 1, depending on which buttons are pressed.

Update view

I chose to make it a bit more visible how fast you are going. I have done this by giving the player a bit leeway based on his speed. When the player is on max speed it will reach about half of the screen. When its speed is about half of the max speed the player will be on the quarter of the screen. This is implemented in the Game class. Here I calculate the position where the player can maximum be based on its speed. If the player is beyond this position the view will move up. All entities that need to be in the view like the lane lines and score will be updated by the same amount as the view.

Events

Window events that are covered are: Resizing and closing the window.

End loop

When a hiker is beyond the finish line it is removed. When a player is removed the game-loop ends, and a new loop begins. Between loops the new score is added to the high scores and the top 10 is written to a text-file. When R is pressed this end-loop ends. The run-function returns true to “main.cpp” and a new game starts.

World

The World class will be used to represent the game world. It will use factories to create entities and keep a list of those entities.

The AI for the enemies is also implemented in world. It will look if there is an obstacle in a given distance above it and if it can move left or right. Based on these inputs it will have a couple options (left, right, yell). It will take randomly a possible action (implemented in the EnemyHiker class). This works well, and I found it quite hard to keep up, so I gave the player a higher maximum speed to compensate.

For the obstacles: A knight will stand still and when yelled at it has 50% chance to switch lanes. A rat will slowly move up, when yelled at it has a 50% chance to double its maximum speed for a given time. Obstacles do not interact whit each other.

For collision detection another class is used, here I give 2 entities and check if they are in collision. Based on how heavy they are one will move farther back then the other. It can also be that an entity is transparent like the rat or text bubble. When a rat is in collision with a hiker that hikers maximum speed will be halved for some time (time lock). Collision is done by watching if the center + half the size of 2 entities intersect and if the lanes are the same.

When a player is in collision or when it yells, it will give the subject a value that updates the observer.

To update the live score the world will call on the observer of the player to get the current score. At the end a bonus will be given: 100 points for each enemy behind the player.

Error detection

There was not much error handling needed. I output an error message if an image/font is not found or if the there was a problem with the scoring. The game will still run but without the high score or with the image/text.

Entity

How the Entity class is build is shown in “Figure 2”. I used the composite state pattern to make the entities via world.

I also used a HikerSFML class that is used to implement functions used by more entities like render and update visuals.

Yelling

When a hiker yells a text bubble will pop up for some time. Only when that text bubble is away, you can yell again.

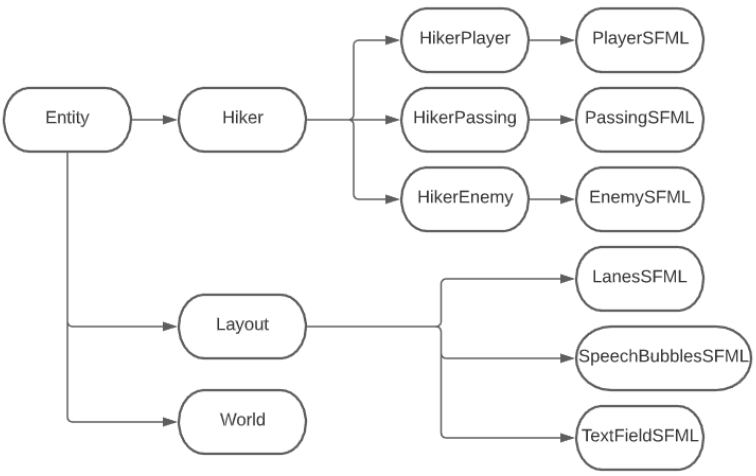

Figure 2 Hierarchy entity