ELEC-A7150 - C++ Programming
Micromachines2

Aarni Halinen, 424974
Akke Luukkonen, 356149
Jan Gustafsson, 424292
Okko Järvinen, 353391

1. Program overview

The program is a racing game in the style of Micro Machines. One or more players can race on various tracks. Players can use various weapons and projectiles to their advantage.

2. Basic functionality

As the user starts the program, a window containing the main menu is opened. The user can choose to start a new game, edit maps, edit settings or quit the program. The user can select the amount of players, the map and the vehicle in the new game menu.

The game starts with the selected settings. The player can accelerate and steer the vehicle through the race track. The player has also selected a weapon of choice, that can be deployed when the user wants. The current standings are displayed in the top left corner of the window along with the lap count and elapsed time. Depending on the track, the user will be penalized for driving outside the designated track. The map contains various special items that may for example give the player a speed boost or blow up the vehicle. The first player to complete the required amount of laps wins. The program now returns to the main menu.

The program settings give the user an option to set key bindings, change the game resolution or alter sound levels.

3. Class structure

As the game starts main creates an instance of Game and Engine that are the actual operators of the game. Main then deletes the Menu and passes control over to the game operators. The split has been done so that Game is the book keeping class and Engine is the class that runs the game. In other words, Game contains all the information of the Objects and it generates or loads an instance of Map, while Engine updates the state of the game.

Engine class does all of the actual work in running the game. It gets all necessary information about the game state from Game and operates accordingly on this information. Engine updates all of the Vehicles positions on the Map and does some calculations e.g. who is in the lead, has a certain Vehicle's speed changed etc. After updating all of this information it passes it back to the Game that updates itself with this information.

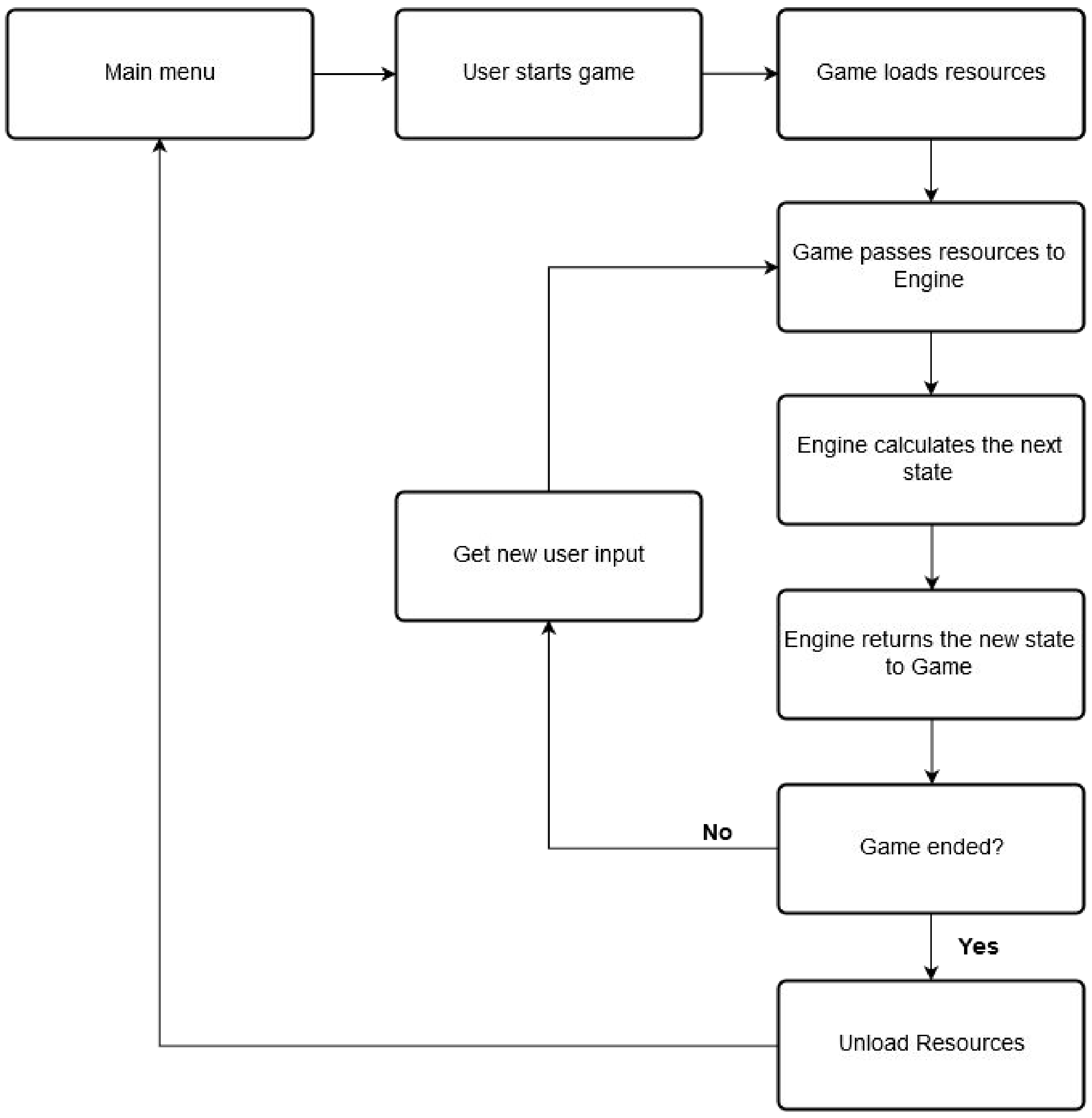
The first duty of the Engine is to update the game state according to the passed time i.e. how much distance a Vehicle has travelled with it's current velocity, has the Vehicle been issued a new heading with a turn command, has the Vehicle accelerated to a new speed and therefore velocity or has the Vehicle collided with something e.g. track boundary, another Player or a Projectile. Finally the Engine will draw the game state to the window.

First it will draw the background i.e. Blocks on the map, all of the Projectiles on the map and finally the Vehicles. This way the Vehicle's texture will always be on top and a player is able to see where he is.

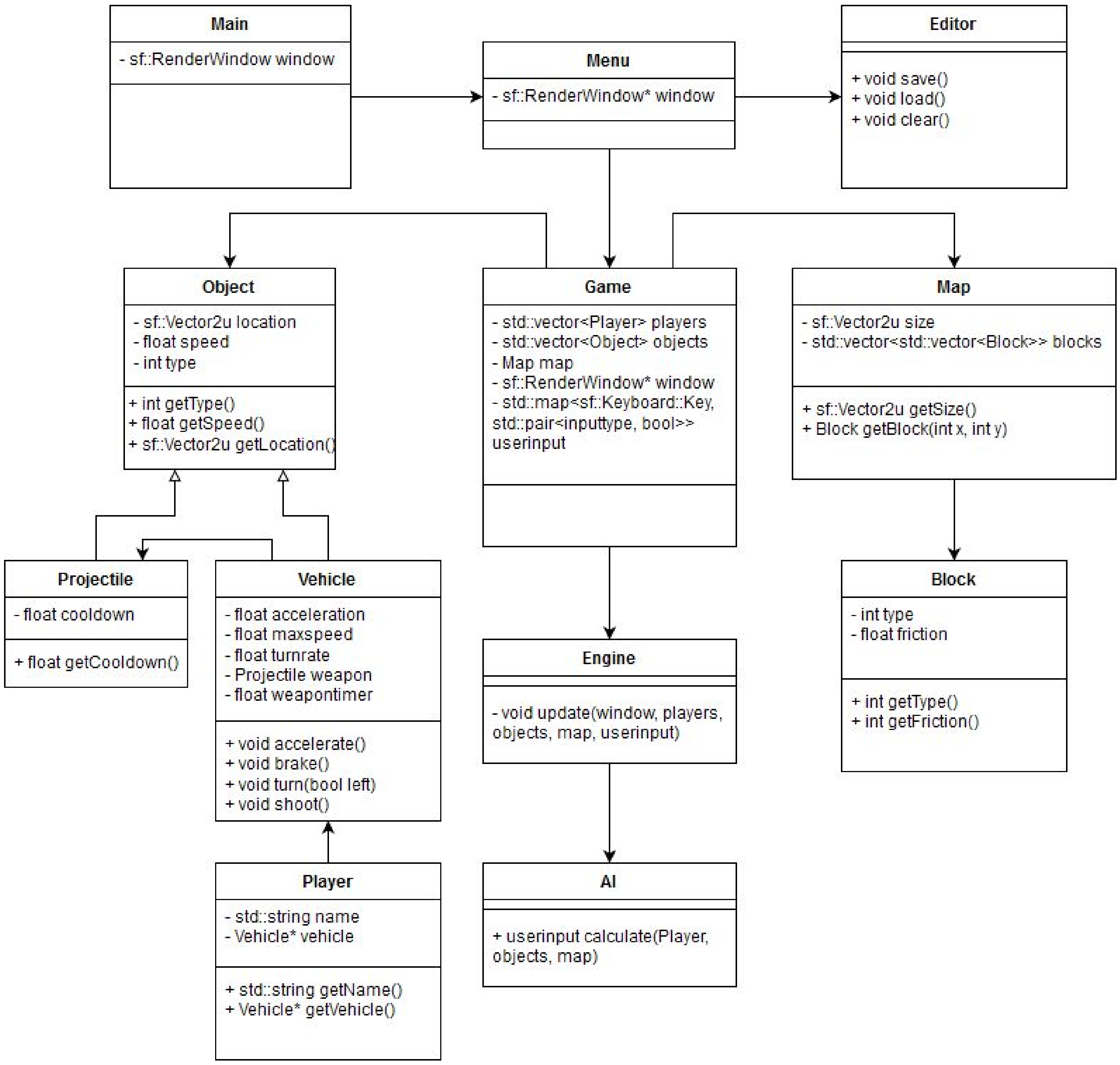
Map contains information about the game tiles called Blocks. Blocks have information about certain types and attributes i.e. friction or can it be driven over. The generated Players travel on these blocks and are affected by their attributes. Hitbox detection will be used to make it impossible for the cars to drive over certain blocks e.g. track boundaries if that is desired in the map. Some of the maps will support driving outside the track boundaries, but punish the player with greatly reduced Vehicle speed or with a "falling off" effect i.e. the Vehicle falls off the track and after a time penalty is re-generated on the track.

The Players control Vehicle classes that have information about their current heading, speed and location. The Player in this case can be a human user or an AI created by the Game. In case of an AI, it will use the same Vehicle as a human user would, but it's user input will be generated by the AI class on request by the Engine. The AI will receive information about the game state and calculate an appropriate response.

A flowchart describing the operation of the program.


Main menu
User starts game
Game loads resources
Game passes resources to Engine
Engine calculates the next state
Get new user input
Engine returns the new state to Game
No
Game ended?
Yes
Unload Resources

4. UML-diagram



Object (base class, public inheritance from sf::Drawable and sf::Transformable)

- location: Location of the object on the Map
- speed: Speed of the object
- type: Type of Object eg. Vehicle type (car, boat), oil spill, mine
- getType(): Returns the type variable

- getSpeed(): Returns the speed variable
- getLocation(): Returns the location variable

Projectile (public inheritance from Object)
- cooldown: Time period how often the Projectile can be deployed
- getCooldown(): Returns the cooldown variable

Vehicle (public inheritance from Object)
- acceleration: Modifier on how quickly the car’s speed changes
- maxspeed: Limit on how large the speed can be
- turnrate: Modifier on how quickly the car can change heading
- weapon: Projectile assigned to Vehicle for example Vehicle can deploy oil spills or mines
- weapontimer: Keeps track that the weapon isn’t deployed until it’s cooldown period has elapsed
- accelerate(): Increases speed by an exponential function using acceleration and maxspeed
- brake(): Decreases speed by an exponential function using acceleration and maxspeed
- turn(bool left): Changes heading according to received parameter and turnrate
- shoot(): Deploys weapon if weapontimer is larger than weapon.cooldown

Player
- name: Player name eg. “Player 1”, “CPU 2”
- vehicle: A pointer to the Vehicle assigned to the Player
- getName(): Returns the name variable
- getVehicle(): Returns the vehicle variable

Engine (static class)
- update(window, objects, map)
- bunch of calculate functions

Block
- type: Type of Block eg. “Empty land”, “Water boundary”
- friction: Modifier that affects a Vehicle’s acceleration
- getType(): Returns the type variable
- getFriction(): Returns the friction variable

Map
- size: 2-dimensional vector with information about the map size (x, y)
- blocks: 2-dimensional matrix containing the Blocks that the map is built of
- getSize(): Returns size variable
- getBlock(int x, int y): Returns Block in the matrix position (x, y)

AI (static class)
- calculate(Player, objects, map): Returns a simulated user input to the engine.

Game
- players: Vector containing all Players
- objects: Vector containing all Objects
- map: An instance of class Map
- window: Pointer to sf::RenderWindow ie. the window for the game to be rendered
- userinput: Map

5. The schedule

The game will be built class by class. The base classes will be built first, followed by more specialized classes. Finally all the features and functionality will be implemented. Tests will be implemented along with functionality.

Week 45
By the end of the week the project plan should be clearly outlined. Additionally, the base structure of the program should be programmed.

Week 46
The goal is to build the map and create some textures for the vehicles and map blocks. With the core of the class structure implemented, the first car should be now driveable. Also the map should be rendered in an appropriate way.

Week 47
By the end of the week basic interaction between objects should be functional. This means that the map boundaries now interact with the vehicle properly. The player centered camera view is implemented next. Here the player stays centered in the window and the map moves around the player.

Week 48
Implementing support for concurrent players is the goal for this week. By now the game should be somewhat playable. The engine is updated to correctly handle multiple players. Projectiles, such as mines and traps, will be implemented next. The engine should also handle all interactions with players and projectiles.

The remaining time will be spent implementing the AI and the map editor, adding content such as new vehicles and weapons, and also refining the engine and playability.