28.10.2020

Project plan - Micro Machines

ELEC-A7151

Elias Nurmi
Hilda Palva
Markus Tuominen
Ville Pihlava
Veeti Kahilainen

Scope of the work

In a nutshell

We are implementing a driving game in which the players can race with little toy cars in everyday-like environments. More than one player can join a game over the internet. The view of the game is top-down.

The most important functionalities are driving physics, obstacles and collisions. The driving physics are implemented with box2d-library. Multiple different tracks can be loaded from files. Different tracks have different terrains and different car handling.

It is also important that there is a lap timer in the game to show the players how fast they drove.

The map is grid-based and 2d. Objects are located in the grid. The view is top-down, each player having their own small view.

There are different obstacles in the track so that it is possible to collide with them. It should not be possible to drive through objects. Objects are either static or mobile.

In addition there are sound-effects in the game which occur for example when a car collides with an object.

When the user starts the program, the main menu asks if the user wants to host the game, join a game or change settings. If the user chooses the join-option, they must enter joining code and then they can choose a car. When all players have chosen their cars, the game will begin.

In the game view there will be a lap timer and position ranking view. In the end of the game there will be a scoreboard which will show results of the game.

If the user chooses host-option from the main menu, he will be able to choose the map, amount of players and car. When these are chosen and everybody joining the game is ready, the game will begin.

In the settings view it is possible to change volume level and player username.

Core features

The features below are required by the project definition or were chosen as “main features”.

Basic features:

- Basic gameplay with simple driving physics and multiple players
    - Driving physics with box2d
    - Lap timer
    - Multiple players by networked multiplayer
    - Multiple tracks loaded from files
    - Grid-based map system
    - 2D maps
    - A couple of test maps
    - Checkpoints
- Game objects which affect gameplay
    - Box2D physics objects
    - solid obstacles
    - oil spills
    - boosts

Additional features:

General:

- Sound effects
    - Car sounds
    - Other vehicle sounds
    - Other game object sounds
    - Collision sounds
    - Simple music
    - Finish line fanfare
- Different car handling on different terrains
    - Asphalt
    - Grass
    - Sand
    - Water
    - Mud
- Different kind of vehicles
    - normal car, f1 car, monstertruck, motorbike
    - choosable color
    - different physics
        - Different grip, acceleration, max speed
- Powerups + Weapons
    - Boosts
    - Banana peels
    - Weapons
- Networked multiplayer

Extra feature ideas

These are additional ideas to be implemented if the time constraints of the project allow it.

- Minimap
- Splitscreen
- Random generated levels
- Race position display
- Artificial intelligence
- Level editor

High-level structure of the software

UI plan

See attachments at the end of the file.

UML chart

See attachments at the end of the file.

Scene system

The game's UI is implemented with a scene system. Scenes are pure virtual classes which are a specific view of the UI. MenuScenes are Scenes that contain SceneComponents which can be buttons, textboxes, pictures and more.

Network system

The game's network system contains a client and a host service. These services are run in separate threads to the main thread. The host service provides necessary data for the clients. The client service provides a communication channel with the client and the game. The first iteration of the network system is supposed to provide clients the ability to see other players, but not interact with them. If enough time is available, interaction with other players will be implemented.

Game logic

Game logic is based on abstract GameObjects, which represent any object (car, tile, obstacle, powerup) in the game world. GameObjects have a transform which represents their position and rotation in the game space and a drawable sprite. GameObject is inherited by Tile and DynamicObject classes. DynamicObject objects have a Box2D collider and are further inherited by cars, pickups, obstacles, etc.

The GameScene class implements the pure virtual class Scene. Input is handled in the HandleEvents function, game logic and physics are handled in the Update function with the box2d library. Drawing is implemented in the Draw function by getting the sprites of all GameObjects in the scene.

The map is represented in JSON format and can be saved to and loaded from a file. The map consists of tiles, checkpoints, and other objects.

Planned use of tools and external libraries

CMake

Makefile generator. Some project members use Linux natively, some use WSL, and others use Windows to develop; CMake makes the workflow more similar for everyone.

Valgrind

Memory debugging tool. Valgrind will be used to track down memory leaks. Preferably all pull requests should be memory leak free.

Box2D

2D physics library. Box2D will be used to simulate the top-down world, car handling, and powerups. All game objects will use Box2D coordinates as their “world space” coordinates. It was chosen because of its simplicity and recommendations.

SFML

Multimedia library. Windows, scenes, UI, sound effects, music, and networking will be implemented using SFML. Chosen over SDL for its simplicity and beginner friendliness.

Division of work and responsibilities between the groups

Ville

- UI planning and implementation
- Networking
    - Client - server interaction
- Game object rendering

Elias

- UI planning
- Sound system planning and implementation
    - Impact sounds, button sounds, acceleration sounds

Markus

- Map system
    - saving
    - loading
    - tiles
    - (generation?)
- Game object rendering

Veeti

- Physics using Box2D
    - World updating
    - GameObject
    - basic car handling
- Theme music

Hilda

- Dynamic objects
    - Boosts
    - Powerup
    - Weapons
    - Obstacles
    - Vehicles (Traditional car, formula, truck etc.)
- Event handling

Planned schedule and milestones before the final deadline of the project

The project has approximately 6 weeks of allotted time for the actual project development and writing documentation. We will divide the workload into 6 rough parts. Each week has separate milestones that we will hopefully achieve.

Week 0:
Development environment working for each team member.
Project plan.

Week 1:
Initial project structure and base class construction.
GameScene, GameObject, and GameObject related classes.

Week 2:
Some version of map is working.
Implementing initial gameplay elements and player controllers.
Networking implementation.
Sound system implementation.

Week 3:
Elements for the flow of the game (races, laps, win conditions) implemented.

Week 4:
Additional features and other extra features could be implemented here, if base features are there to support them.

Week 5:
Project documentation started.
Finishing core features by this point latest.
Code should be cleaned of possible bugs at this point, not later.

Week 6:
Project documentation finishing touches.
Project code polishing.

SceneManager
- scenes_: std::map<std::string, Scene*> scenes_
- currentScene_: Scene*
+ AddScene(std::string, Scene*): void
+ ChangeScene(std::string): void
+ GetCurrentScene(): Scene*
+ HandleEvents(sf::RenderWindow&): void
+ Update(const sf::Time&): void
+ Draw(sf::RenderWindow&): void
HostService
- listener_: sf::TcpListener
- selector_: sf::SocketSelector
- clients_: std::list<sf::TcpSocket*>
- running_: std::atomic<bool>
+ Listen(unsigned short): sf::Socket::Status
+ Start(): void
+ Stop(): void
+ IsRunning(): bool
- SendToAll(sf::Packet&): void
- SendToOne(sf>>TcpSocket*, sf::Packet&): void
ClientService
- socket_: sf::TcpSocket
- selector_: sf::SocketSelector
- running_: std::atomic<bool>
- packetQueue_: std::deque<sf::Packet*>
- messageFunctions_: std::map<std::string, std::function<void(sf::Packet&)>>
- packetQueueMutex_: std::mutex
+ Connect(const sf::IpAddress&, unsigned short, sf::Time): sf::Socket::Status
+ Start(): void
+ Stop(): void
+ IsRunning(): bool
+ Send(sf::Packet*): void
+ AddMessageFunction(const std::string&, std::function<void(sf::Packet&)>): void
- ReceiveWithTimeout(sf::Packet&, sf::Time): sf::Socket::Status
<<abstract>>
Scene
+ HandleEvents(sf::RenderWindow&) = 0: void
+ Update(const sf::Time&) = 0: void
+ Draw(sf::RenderWindow&) = 0: void
MenuScene
- sceneComponents_: std::vector<SceneComponent*>
+ HandleEvents(sf::RenderWindow&): void
+ Update(const sf::Time&): void
+ Draw(sf::RenderWindow&): void
+ AddSceneComponent(SceneComponent*): void
GameScene
- gameObjects_: std::vector<GameObject>
- box2DWorld_: b2World
- map_: Map
<<abstract>>
SceneComponent
- relativePosition_: sf::Vector2f
- relativeSize_: sf::Vector2f
+ HandleEvent(sf::Event&, sf::RenderWindow&) = 0
+ virtual Update(const sf::Time&) = 0: void
+ virtual Draw(sf::RenderWindow&) = 0: void
Map
- tiles_: std::vector<Tile>
+ loadMap(filename): type
GameObject
- sprite_: sf::Sprite
- transform_: Box2d::Transform
+
Extends
Extends
Tile
+ material: TileMaterial
+ method(type): type
DynamicObject
+ collider: Collider
- isStatic_: Boolean
+ method(type): type
Extends
Extends
Extends
Extends
Car
- Box2dThings_
Obstacle
+ type_: struct obstacle
+ getType() : obstacle
Powerup
+ field: type
+ method(type): type
Weapon
+ type_: struct weapon
+ shoot(direction) : Bool
TableSceneComponent
- rectangleShape_: sf::RectangleShape
- text_: sf::Text
- character_: sf::Text
- columnCharacterLimits_: std::vector<int>
- rowLimit_: int
- rowCharacterLimit_: int
- table_: std::deque<std::vector<std::string>>
+ AddRow(const std::vector<std::string>&): int
+ RemoveIndex(int): void
+ ReplaceIndex(int, const std::vector<std::string>&): void
- SetPosition(const sf::Vector2f&): void
- SetSize(const sf::Vector2f&): void
- IsMouseHovering(sf::RenderWindow&): bool
- UpdateText(): void
TextInputSceneComponent
- backgroundColor_: sf::Color
- highlightColor_: sf::Color
- rectangleShape_: sf::RectangleShape
- text_: sf::Text
- textString_: std::string
- unsubmittedTextString_: std::string
- selected_: bool
- characterLimit_: int
- cursorPosition_: int
- cursor_: sf::Text
- onSubmit_: std::function<std::string(std::string)>
- SetPosition(const sf::Vector2f&): void
- SetSize(const sf::Vector2f&): void
- IsMouseHovering(sf::RenderWindow&): bool
PictureSceneComponent
- texture_: sf::Texture
- sprite_: sf::Sprite
- SetPosition(const sf::Vector2f&): void
- SetScale(const sf::Vector2f&): void
ButtonSceneComponent
- backgroundColor_: sf::Color
- highlightColor_: sf::Color
- rectangleShape_: sf::RectangleShape
- text_: sf::Text
- onClick_: std::function<void()>
- buttonSound_: sf::Sound
- buttonSoundBuff_: sf::SoundBuffer
- SetPosition(const sf::Vector2f&): void
- SetSize(const sf::Vector2f&): void
- IsMouseHovering(sf::RenderWindow&): bool
TextSceneComponent
- text_: sf::Text
- SetPosition(const sf::Vector2f&, const sf::Vector2f&): void
- SetSize(const sf::Vector2f&): void

MAIN MENU
2D Car Game
Host
Join
Settings
HOST
BACK
Lobby:
<SELECTED MAP>
1. PLAYER1 <car> <ready?>
2. PLAYER2 <car> <ready?>
SELECT CAR
SETTINGS (ONLY OWNER)
text was inserted here
READY/START GAME
insert text here
LOBBY SETTINGS
BACK
SETTINGS
<laps>
select laps
<selected map>
select map
<max players>
select max players
<selected vehicles>
select vehicles
GAME SCREEN
<THE GAME>
<minimap>
Lap time: 11.70s Lap: 1/3 Speed: 40km/h Pos: 1/8
GAME OVER SCREEN
Pos Name Time Best Lap
1. Player3 3:31 0:52
2. carman 3:54 0:53
BACK
JOIN
BACK
Enter IP:
<ip enter box>
submit
SETTINGS
BACK
SETTINGS
volume
Player
username
SELECT CAR
BACK
SELECT CAR
<pic of car>
<selected car>
(dropdown menu)
<SPECS>
<selected color>
(dropdown menu)
ESC MENU
BACK
QUIT