ELEC-A7150 - C++ Programming
Micromachines2

Aarni Halinen, 424974
Akke Luukkonen, 356149
Jan Gustafsson, 424929
Okko Järvinen, 353391

1. Program overview

The program is a racing game in the style of Micro Machines. The game supports up to four players in split screen mode and the game provides a map editor for creating custom maps. The players are also able to use mines to their advantage during the race. The mines cause the vehicle to launch backwards. The program also randomly generates oil spills on the track. Driving into an oil puddle will make steering impossible.

As the program starts, a main menu is opened. In this menu the user can set the amount of players, select a map or start the map editor. The map editor features five different surface materials and the user can save and load maps.

2. Software structure

The program is composed of following classes and namespaces: AI, Block, Config, Editor, Engine, Game, Hitbox, Map, Menu, Object, Pausemenu, Player, Projectile, ResourceManager and Vehicle. Additionally, the program utilizes several text files that define information regarding textures and sounds.

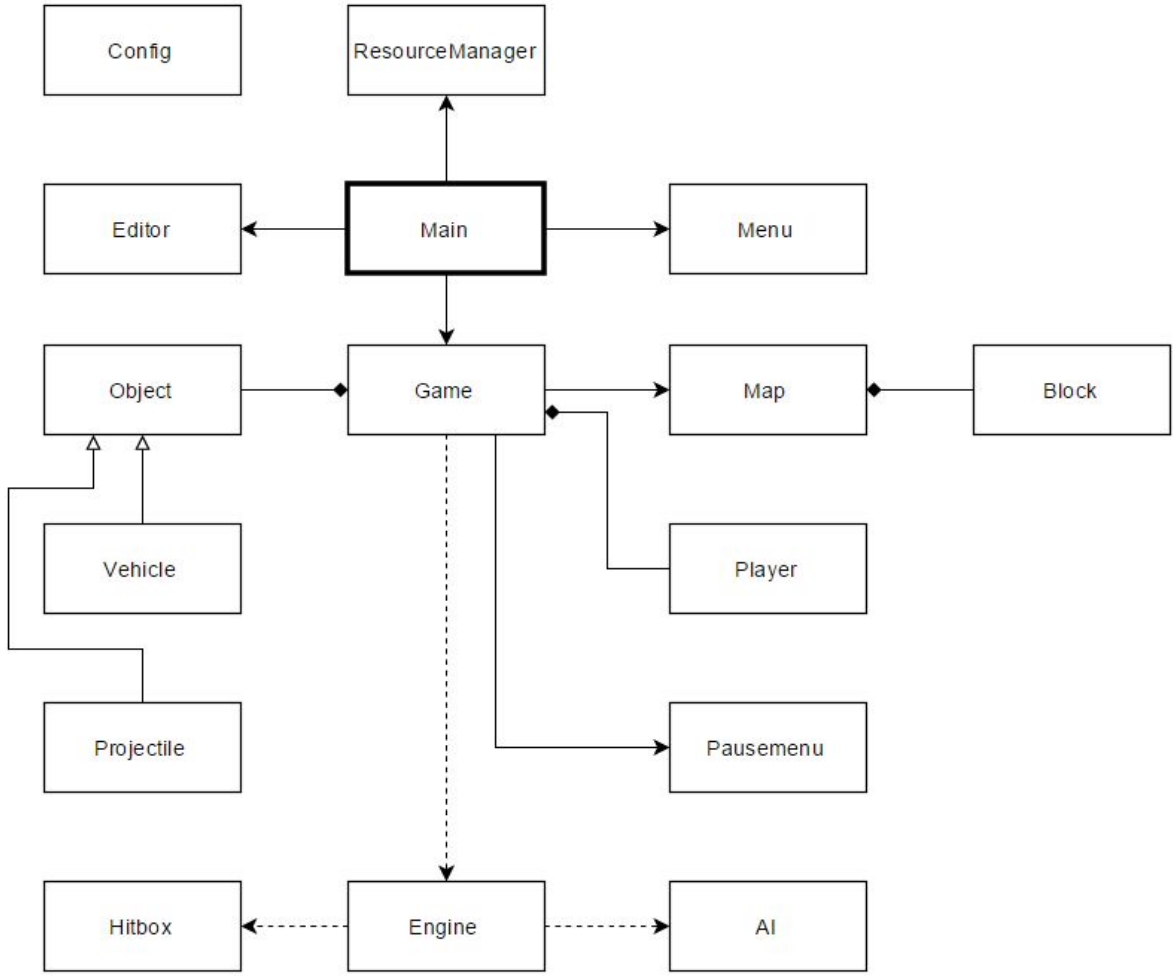
The main function initializes the ResourceManager, which loads all the textures and sounds needed to run the program. The main function then runs a screen handler loop that determines which part of the program is to be displayed. Initially, the main menu will be shown. Depending on the user's choices in the main menu, either the map editor or the game view is opened. These options call the Editor class and the Game class respectively.

The Game class runs the event loop of the game and it calls the Engine to draw each cycle and move the objects. Game also initializes all the objects and players in the game based on the information passed by the Menu. The Game class passes user input- and object -data to the Engine on each event loop cycle. The Engine namespace contains multiple functions for determining the next game state based on user input. Additionally, the Engine renders the split screen view based on the number of players.

The objects in the game (Vehicle and Projectile) are based on the Object class and the race track is an instance of the Map class, which is composed of Block instances. The Player class contains player specific information, such as name, and a pointer to the player's Vehicle. Each Vehicle has it's own Projectile to be used as a weapon.

The Config namespace contains defining information about Blocks, Objects and user input. Additionally, the class contains several maps that link for example a type of block to it's respective friction value. These are used mostly in game initialization. The Hitbox namespace is used in checking object interactions in the game. It contains functions for creating hitboxes for objects and checking collisions between objects.

The class structure of the program



3. Software logic

Game::run() -function runs the game loop. User input is gathered here into pairs, which contain a pointer to the player that gave the input and the type of the input (the pressed key). Additionally, this function takes care of the timing of the program by measuring elapsed time on each cycle and later passing it on to the Engine. At the end of each cycle, the function checks whether a player has crossed the finish line and if the race has ended. Depending on the situation, the function will show a winning screen. If escape is pressed during the race, a pause menu is shown. This function is called only when the game starts initially.

Engine::update() is the main function of Engine namespace, that handles calls of other engine functions. It takes data from Game class as pointers and calls functions that for example move game objects. Engine handles movement by calling object's' own move functions, takes care of collision detection and drawing of current tick. Also lap counting for individual vehicles is done in engine. Engine does not return anything.

Editor can be accessed from the main menu and it works separately from the rest of the program. The Editor works by modifying two images, block_image and image. Block_image contains information about the BlockTypes and image contains what the user is seeing. Using the brushes to modify the image, the user actually modifies both images and the block_image is the only one that is actually saved to the disk when saving. The brushes work by applying image masks to the images and then adding The map then is rendered based on this. Editor currently pre-renders the map image during saving for faster loading times. After closing the editor, the user it returned back to the main menu.

4. Instructions for building and using the software

Building the project

A makefile for building the software with GNU compiler collection (GCC) is included in the project folder. The software uses SFML library version 2.3.2, so linker have to be set up accordingly. Used SFML libraries are: **sfml-audio.lib**, **sfml-graphics.lib, sfml-window.lib** and **sfml-system.lib**. If building with Visual C++, **sfml-main.lib** is also needed for compiling.

Due to the use of SFML-2.3.2, GCC version 4.9 or above is required. SFML files are not included in the project, so these must be installed before building the project. Most package managers have old version of SFML, so installing from website is probably the best choice. After install is done, modify Makefile macros “INC_PATH” and “LIB_PATH” to point to SFML’s include and lib folders.

NOTE: Due to some cross-platform related bug, cars in-game possibly do not turn when given input on Linux OS. Functionality of the software is not guaranteed on Linux.

On Windows OS, building the project with Visual Studio requires include and library paths to be specified for the project. This settings are found under project properties: **C/C++ -> General -> Additional Include Directories**, **Linker ->General-> Additional Library Directories** and **Linker -> Input -> Additional Dependencies**. For debug configuration, use sfml-xxx-d.lib files for additional dependencies instead.

Using the software

On Linux, one can start the program by changing directory to project root after building, and calling “./main”, or “export LD_LIBRARY_PATH=<sfml-install-path>/lib && ./main” if libraries are not installed to OS default location.

On Windows, project_folder/resources have to be copied along with .DLL files installed in SFML-2.3.2/bin folder have to be copied to project_folder/Release. After files are copied, run the executable.

Game controls (accelerate, brake, turn left, turn right, drop mine):

1. Player: W, S, A, D, Left Ctrl
2. Player: Up Arrow, Down Arrow, Left Arrow, Right Arrow, Right Ctrl
3. Player: Numpad8, Numpad5, Numpad4, Numpad6, Numpad0 (**NOTE: use Num lock**)
4. Player: I, K, J, L, Space

Controls are based on number of player, ie. if player 1 is off and other 3 are in use, player 2 uses 1. player controls and so on.

Game play:
First player to complete 3 laps is the winner. Watch out for oil spills and other player mines and stay on track to win!

Editor usage:
The editor first asks whether to create a new map or edit an existing one. By typing ‘new’ a new map is created. Next the editor asks for the desired map size ranging from 12 to 30 (blocks of 256 by 256 pixels). Finally the user can choose a base block for the map. In the editor, F1 opens instructions, number keys from 1 to 5 change the current block, e and q change the brush size and the drawing is done with mouse left click. The map can be saved by pressing s.

5. Testing

Testing was mostly done with the debug environment included in Visual Studio. After writing or modifying a function, the debugger was launched with breakpoints and functionality was manually inspected step-by-step by observing the changes in variables and execution paths of the code.

A typical situation could be in the form of implementing the load functionality for Map.
The code read a image file from the disk and built usable game data from it. Checks were manually done to see that the code extracted the correct information and handled it properly.

- Loading the image from disk.
- Extracting the correct amount of bytes from the image.
- Conversion of the block information from the image color data to useful BlockType information.
- Successful loading of the pre-rendered image or rendering of the image based on this BlockType information.
- Reporting if the load was a success or failure.

This workflow was used by every project member systematically.

6. Work log

Work plan:
Initial plan was to give a part of program to each member and implement the program in parts. Once a part was considered finished, a new part was assigned to the member. Interfaces between parts were usually worked together once both parts were almost finished. The goal was to efficiently develop different parts of the program simultaneously. Occasionally, we worked together on some more problematic parts of the program. Communication was done using Mumble and Facebook chat group. Responsibilities can be seen from individual worklog.

Individual worklog:

Week 1:
Aarni: 4h
- Project planning

Akke: 4h
- Project planning

Jan: 4h
- Project planning

Okko: 4h
- Worked on project plan

Week 2:
Aarni: 12h
- Engine namespace implementation

Akke: 8h
- Some of the class files created. Began working on Map implementation.

Jan: 1h
- Menu design in paper

Okko: 12h
- Started the Game class implementation
- Created main.cpp

Week 3:
Aarni: 4h
- Menu class development start. Because leaving for a trip, menu left for Jan to develop

Akke: 15h
- Map implementations and basics of Editor added.

Jan: 2h
- Created all menu icons

Okko: 16h

- Continued implementing Game (initializing functions and definition files)
- Created some car textures and block textures

Week 4:
Aarni: 8h
- Linux compiling and Make

Akke: 10h
- New features to Editor and optimizing of existing functions.

Jan: 8h
- Menu icons added to git
- First code development for menus

Okko: 16h
- Implemented ResourceManager
- Implemented tickrate system for the Game loop
- Added the theme song and higher resolution textures
- Linked the map to the Game and implemented friction based movement for the vehicles
- Implemented split screen view and added a mine projectile

Week 5:
Aarni: 14h
- AI development

Akke: 12h
- Hitboxes and basic collision detection added.

Jan: 18h
- First buildable menu added to project

Okko: 20h
- Implemented shooting for vehicles
- Implemented a sound manager
- Developed the split screen further and tweaked the vehicle movement

Week 6:
Aarni: 35h
- AI development continue. Lots of bugfixes, features: lap counting, 4-player split screen multiplayer
- Project documentation

Akke: 20h
- Fixes here and there. Internal changes to Map for even more further optimizations.

Jan: 20h
- Finished main menu
- created pause menu
- created all new icons to buttons
- created button class to replace struct button structure, but texture handles lost sprite textures so pause menu have to build with struct button.
- project documentation

Okko: 30h

- Added new cars, oil spills, explosions and the game time counter
- Implemented collision detection for projectiles and walls
- Tweaked sounds
- Tweaked the vehicle movement
- Implemented random generated oil spills
- Cleaned up the code and fixed some bugs