ELEC-A7150 - C++ Programming
Game Implementation: Micro Machines
mmachines2

Arto Lehisto
Sampsa Hyvämäki
Tuomas Poutanen
Mikko Murhu

December 19, 2016

Contents

1 Overview **3**

2 Software structure **8**

3 Description of software logic **9**

4 Instructions for building and using **10**

5 Testing **10**

6 Work log **11**

7 Doxygen **13**

1 Overview

The outcome of the project is a driving game mimicking the style of Micro Machines. The game has a local split-screen two player mode, in which the players compete to finish a set number of laps first. The players have to avoid obstacles while trying their best to disturb their opponent's driving by shooting at him. Launching the game lets the user access the menu. From the menu, the user can change settings or start a game at a specific map. There are multiple maps to choose from and implementing more isn't difficult. Figures 1, 2, 3, 4 and 5 show the views the user sees when navigating in the menu. Figure 6 shows the in-game view.

Figure 1: View of the main menu.

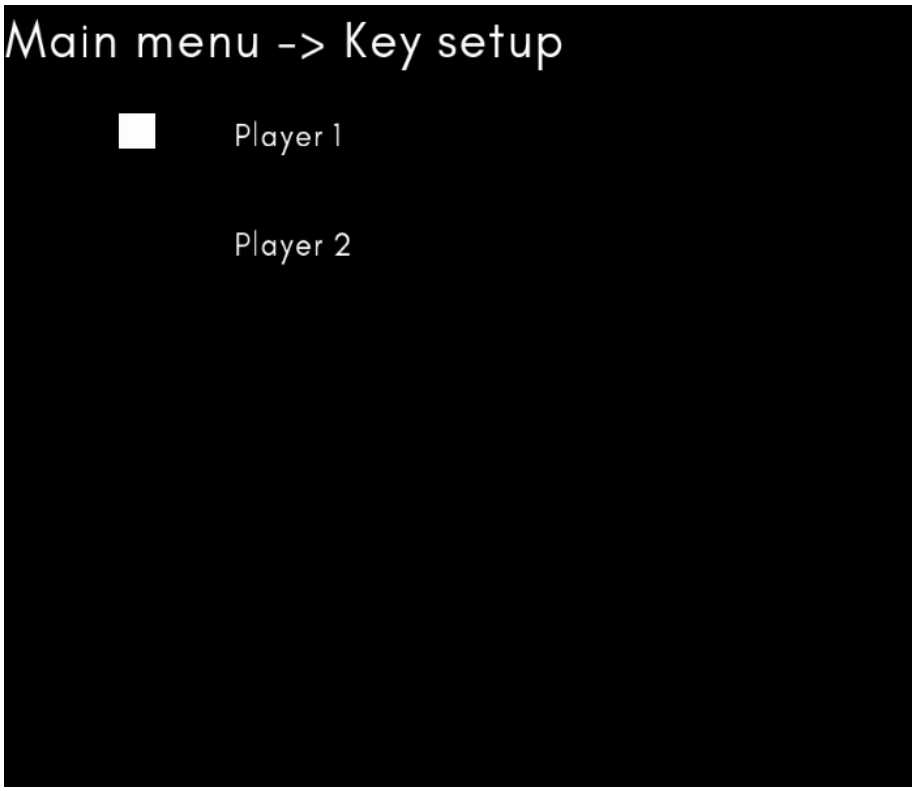


Figure 2: View of the key setup menu.

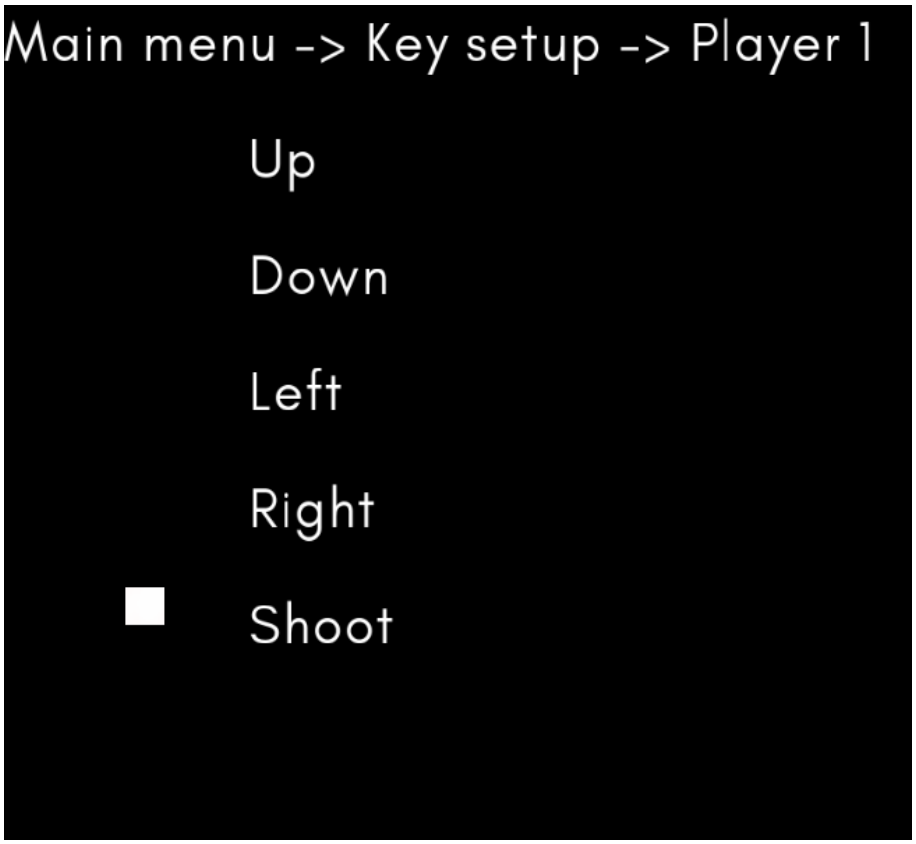


Figure 3: View of the key setup menu, when setting keys.

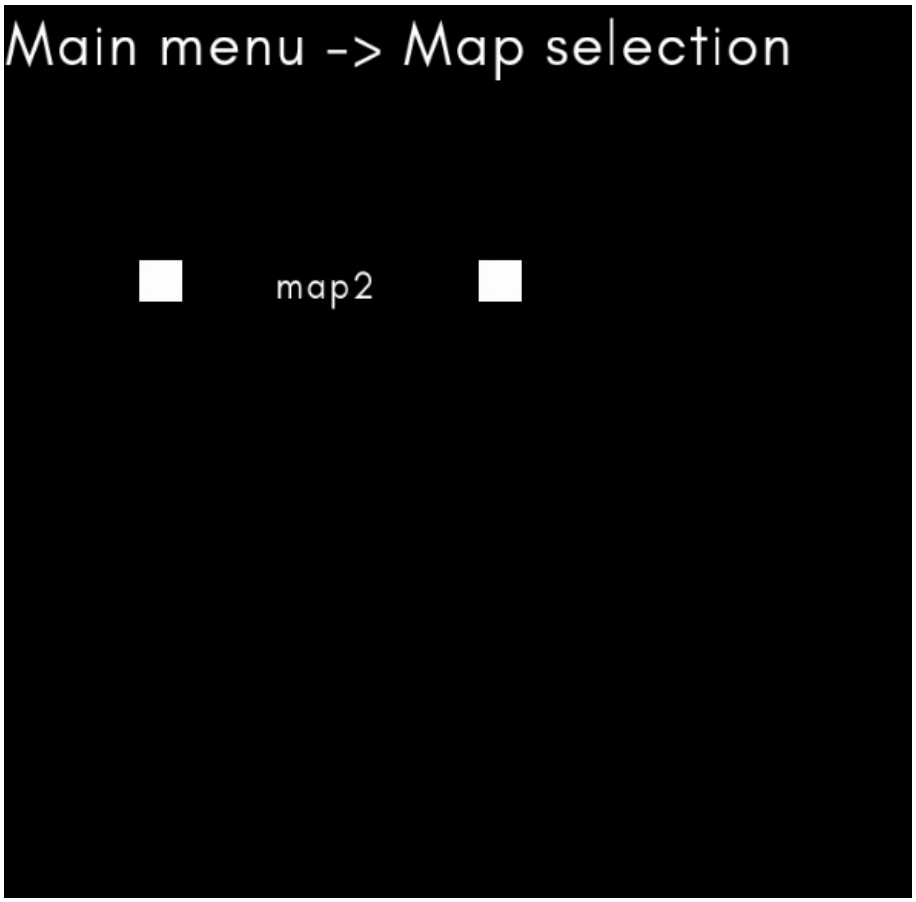


Figure 4: View of the map selection menu. The selection is indicated by the white square on the right-hand-side.

Figure 5: View of the map selection menu. There is no white square next to the map now, so this is not the selected map.

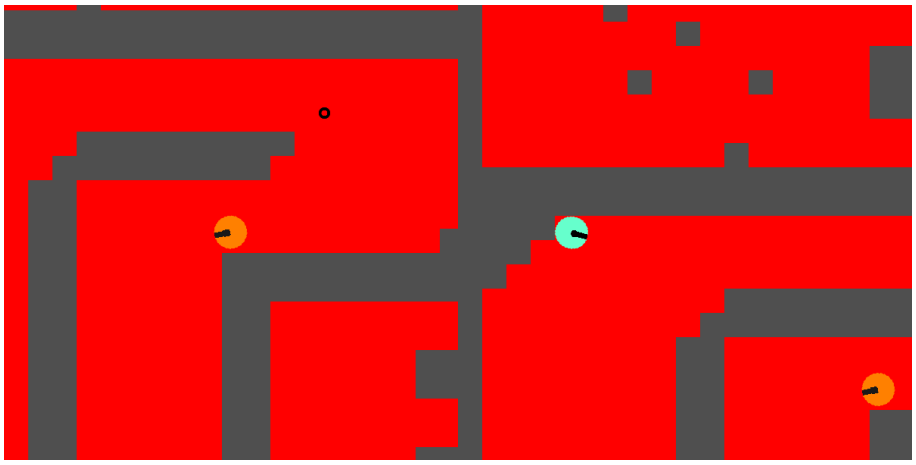


Figure 6: View of the in-game screen. The users control one of the circles that their splitscreen views follow.

2 Software structure

Among with standard C++11-libraries, the software uses SFML (simple and fast multimedia library) for creating a graphical user interface typical for a computer game: while running, the program handles screens, in which the user can define game setting or play the game. The program flow consists of changing back and forth between the screens until the user decides to exit the game. The program is implemented using object-oriented programming: inside the main-function, class instances are created and the values of their member variables are changed by calling their member functions. During the game development, a class structure described in figure 7 was created.

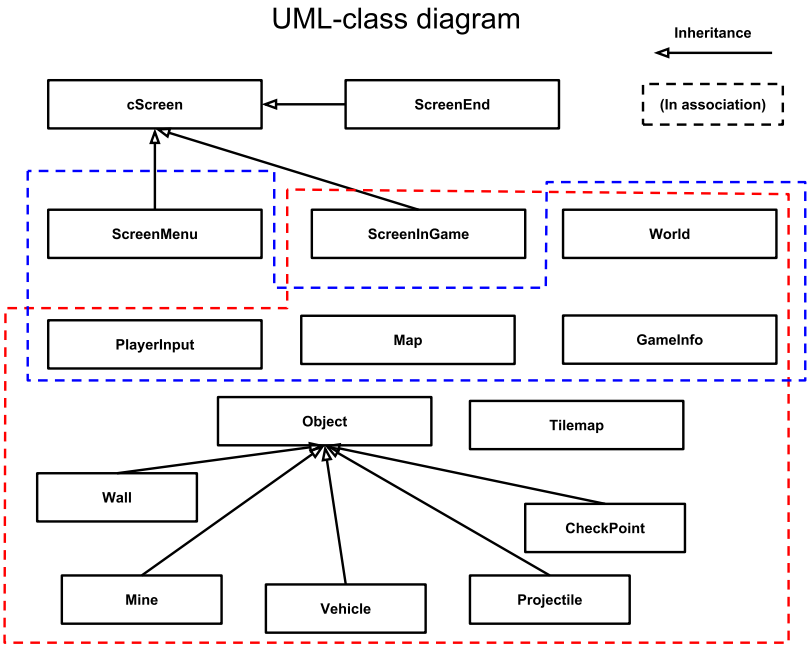


Figure 7: The UML-class diagram of the project.

3 Description of software logic

The game consists of drawable items that move according to user-input and collide resulting in different reactions. The main loop calls for different screens for different game stages. The screens implement while-loops that render the game and in the in-game screen update the status of drawables.

Preserving game data and running relevant checks of collisions and game logic etc. is performed in an instance of the World class. The game map is read from a textfile including locations of walls and checkpoints and parameters of vehicles and mines. The map on the background is created only once in the form of a tilemap class, which is lighter to render than tiles on their own. When the text file is read, map, walls, checkpoints, mines and vehicle parameters are saved in the world instance. Two player class instances are created for controlling player input. The game is ready to start: for each rendered frame multiple physics updates are performed. Each physics update updates the state of the world; that is, location of projectiles and vehicles, the state of game obstacles,

collisions etc.

All collidable items are inherited from the base class Object. The Object-class contains methods that are used by all game objects, such as those linked to collision detection and object rendering.

In addition to the actual gameplay, there are two stages of the game. In the menu, user can define game-settings (choose a map and set input keys) and in the endscreen, player is informed of completing the game. As mentioned before, these stages are implemented by using the screen-class. The class-instance itself has only one member function (Run), which contains instructions to run that stage of the game and, most importantly, draw things to a window given to it as a function parameter. The return value of the function defines the next game stage and indicates the closing of the program. Screen has one member variable, pointer to GameInfo-object which stores important information that is accessible to every screen (e.g. chosen map).

4 Instructions for building and using

We use make for building our project. In order to build the project, one must have make installed on their system. After that one can build and run our project by typing "make run" into a terminal that is in the project's root directory. This builds the project and starts the application. The application consists of two main parts: A main menu and the actual game. The user can navigate in the menu using arrow keys. Up and down arrow keys change the option that would be selected, right arrow key selects the option and left arrow key returns to the previous menu. To select a map, choose the "Choose map" option and scroll through the maps using up and down arrow keys, then select the map with the right arrow key. The selected map will be indicated by a white square to the right-hand-side of the map. After the preferred map is selected, the user can start the game by selecting "Start game" on the top menu.

In game, the user can move using the arrow keys for player 1 and wasd for player 2, by default. Player 1 shoots with space and player 2 shoots with tab. These inputs can be changed in the settings. The user can also press escape to return to the menu.

To quit the game, choose "Quit Game" in the top menu.

5 Testing

In the beginning, testing was done purely by compiling the project to make sure there is nothing wrong with the syntax. We built a main function quite early, and after that we could also test the application by running it. This

kind of testing was mostly done by two methods, by printing something to the standard output to see a glimpse of individual behavior of some function and by taking notice on what happens on the game window and deducting something from that behavior. In case of memory leaks, we used valgrind to find out the offending piece of code.

6 Work log

Week 45: The first implementation of makefile, planning of the project started

Arto: Implemented the makefile, tinkered with git file filtering (8h)
Sampsa: (4 hrs)
Tuomas: Some practice on makefiles (4 hrs)
Mikko: (4 hrs)

Week 46: Project plan was created together

Sampsa: Began the implementation of Map-class, created a function to read the data from textfile to the program memory (8 hrs)
Tuomas: Created a class for Tiles with enumeration for tiletype and function, which later proved to be useless. Watched and read lots of tutorials regarding creating games with sfml (5h)
Arto: Worked on the tile class, worked on the object class (5h)
Mikko: Worked on basic physics, Object-class and Vehicle-class. (6h)

Week 47:

Sampsa: Finished a first working version of map-class, the program could now get the name of the map, tiletypes, dimensions and map matrix from a textfile. Made also first testmaps. (10 hrs)
Tuomas: Learned how to draw tiles. Added basic graphs for tiles to be drawn and figured out how to present the routes for graphics. Had some issues with the scope of the program, since the graph route is different depending on from which context the program is run. (5h)
Arto: Added a Player (later renamed to PlayerInput) class for handling player input. Found out with other team members that separate tiles are way too resource intensive to implement our map properly. Implemented a tilemap class based on sfml's vertex arrays to alleviate this. Moved function implementations from map header file to map code file. (10h)
Mikko: Continued implementing physics, the Object-class and the Vehicle-class. (4h)

Week 48:

Sampsa: In a refresher training (military) from 26.11. to 5.12. (0 hrs)
Tuomas: Implemented splitscreen gameplay with views. (4h)
Arto: More work with the tilemap class, cleaned up main. (2-4h)
Mikko: Began working on the World-class and collision-detection. (4h)

Week 49:

Sampsa: Added gameinfo-class to store game settings and other data between screens. Added other test map and functionality to choose in menu screen which one of the two maps to play. (8 hrs)
Tuomas: Implemented the screens classes to maintain information of game stage. (6h)
Arto: More work with map and tilemap classes. Finally removed tile class and references to it. renamed player class to playerinput class. (5h)
Mikko: Changed the world-class and the main-function to use smart-pointers to avoid problems caused by circular reference of the world- and object-classes. Worked with collision detection and reactions. (4h)

Week 50:

Sampsa: Made checkpoint-class so that vehicles could follow player's progress of the game. Made checkpoints to be loaded from textfile, and implemented functions to vehicle-class to react to the collision with a checkpoint. Added member variables for vehicle-class to describe player's progress. Worked in own branch, and spend a lot of time before figuring out how simple the implementation could be. (12 hrs)
Tuomas: Implemented a new map format for reading more map-dependent data from file including checkpoints, mines and vehicle info. Implemented mines and all collision handling regarding mine collision. Attempted to move all texture data to a different header file but couldn't get it working. Wasted 3 hours of precious code time. Did a lot of testing with the game. (15h)
Arto: Added and configured Doxygen to our project for easy and clear documentation of each of the classes and functions in our project. Started the big task of documenting everything. Cleaned up world creation from files, mainly wall creation. Added functionality for non-hardcoded maps. Redid map file hierarchy. Made a main menu for the selection of player keys and a map. Expanded the gameinfo class that shared data between different game states. Added exception handling and controlled shutdown in case of errors to map loading. (20h)
Mikko: Changed the general case of collision detection to take only distance in account, making it a lot smoother. Implemented the collision detection and reaction for walls separately, making them a lot smoother as well. Added a time-variable to the physics engine, making it possible to have timesteps smaller than 1. Separated the physics-engine from drawing, as in, making a different amount of updates on physical state and draw commands. Created the basis for maps that are used in the project now. Lots of testing and cleaning up the code. (20h)

7 Doxygen

The Doxygen-output for our project follows.

Mmachines2

Generated by Doxygen 1.8.12

Contents

1 Hierarchical Index **1**

1.1 Class Hierarchy 1

2 Class Index **3**

2.1 Class List 3

3 Class Documentation **5**

3.1 CheckPoint Class Reference 5

3.1.1 Detailed Description 5

3.1.2 Constructor & Destructor Documentation 5

3.1.2.1 CheckPoint() [1/2] 5

3.1.2.2 CheckPoint() [2/2] 6

3.1.3 Member Function Documentation 6

3.1.3.1 collide() 6

3.1.3.2 getHeight() 6

3.1.3.3 getNumber() 6

3.1.3.4 getWidth() 6

3.2 cScreen Class Reference 7

3.2.1 Detailed Description 7

3.2.2 Constructor & Destructor Documentation 7

3.2.2.1 cScreen() 7

3.2.3 Member Function Documentation 7

3.2.3.1 get_infoptr() 7

3.2.3.2 Run() 7

3.2.3.3 set_infoptr() . . . 8
3.3 GameInfo Class Reference . . . 8
3.3.1 Detailed Description . . . 8
3.3.2 Constructor & Destructor Documentation . . . 8
3.3.2.1 GameInfo() . . . 8
3.3.3 Member Function Documentation . . . 8
3.3.3.1 get_lap_total() . . . 8
3.3.3.2 get_map_path() . . . 9
3.3.3.3 get_tilemap_path() . . . 9
3.3.3.4 getPlayerInput() . . . 9
3.3.3.5 set_lap_total() . . . 9
3.3.3.6 set_map_path() . . . 9
3.3.3.7 set_tilemap_path() . . . 9
3.3.3.8 setPlayerInput() . . . 9
3.4 Map Class Reference . . . 10
3.4.1 Detailed Description . . . 10
3.4.2 Constructor & Destructor Documentation . . . 10
3.4.2.1 Map() . . . 10
3.4.3 Member Function Documentation . . . 10
3.4.3.1 createWalls() . . . 10
3.4.3.2 draw() . . . 10
3.4.3.3 getCheckPoints() . . . 11
3.4.3.4 getHeight() . . . 11
3.4.3.5 getMines() . . . 11
3.4.3.6 getTilemap() . . . 11
3.4.3.7 getTileSize() . . . 11
3.4.3.8 getWidth() . . . 11
3.4.3.9 loadTileMap() . . . 11
3.4.3.10 setTilesetPath() . . . 12
3.5 Menu Struct Reference . . . 12

3.5.1 Detailed Description . . . 12
3.6 Mine Class Reference . . . 12
3.6.1 Detailed Description . . . 13
3.6.2 Constructor & Destructor Documentation . . . 13
3.6.2.1 Mine() . . . 13
3.6.3 Member Function Documentation . . . 13
3.6.3.1 collide() . . . 13
3.6.3.2 detonate() . . . 13
3.6.3.3 draw() . . . 14
3.6.3.4 getClock() . . . 14
3.6.3.5 isVisible() . . . 14
3.6.3.6 setInvisible() . . . 14
3.6.3.7 setTexture() . . . 14
3.6.3.8 updateState() . . . 14
3.7 Object Class Reference . . . 15
3.7.1 Detailed Description . . . 15
3.7.2 Member Function Documentation . . . 16
3.7.2.1 collides() [1/3] . . . 16
3.7.2.2 collides() [2/3] . . . 16
3.7.2.3 collides() [3/3] . . . 16
3.7.2.4 draw() . . . 16
3.7.2.5 getAngle() . . . 16
3.7.2.6 getMass() . . . 16
3.7.2.7 getPosition() . . . 16
3.7.2.8 getRadius() . . . 17
3.7.2.9 getSpeed() . . . 17
3.7.2.10 getSprite() . . . 17
3.7.2.11 setAngle() . . . 17
3.7.2.12 setMass() . . . 17
3.7.2.13 setPosition() . . . 17

3.7.2.14 setSpeed() . . . 17
3.7.2.15 setTexture() . . . 18
3.7.3 Member Data Documentation . . . 18
3.7.3.1 angle . . . 18
3.7.3.2 mass . . . 18
3.7.3.3 position . . . 18
3.7.3.4 radius . . . 18
3.7.3.5 speed . . . 18
3.7.3.6 sprite . . . 18
3.7.3.7 texture . . . 19
3.8 PlayerInput Class Reference . . . 19
3.8.1 Detailed Description . . . 19
3.8.2 Member Function Documentation . . . 19
3.8.2.1 Down() . . . 19
3.8.2.2 getKeyDown() . . . 20
3.8.2.3 getKeyLeft() . . . 20
3.8.2.4 getKeyRight() . . . 20
3.8.2.5 getKeyShoot() . . . 20
3.8.2.6 getKeyUp() . . . 20
3.8.2.7 Left() . . . 20
3.8.2.8 Right() . . . 20
3.8.2.9 setKeyDown() . . . 20
3.8.2.10 setKeyLeft() . . . 21
3.8.2.11 setKeyRight() . . . 21
3.8.2.12 setKeyShoot() . . . 21
3.8.2.13 setKeyUp() . . . 21
3.8.2.14 Shoot() . . . 21
3.8.2.15 Up() . . . 21
3.9 Projectile Class Reference . . . 22
3.9.1 Detailed Description . . . 22

3.9.2 Constructor & Destructor Documentation . . . 22
3.9.2.1 Projectile() . . . 22
3.9.3 Member Function Documentation . . . 22
3.9.3.1 collide() [1/2] . . . 22
3.9.3.2 collide() [2/2] . . . 23
3.9.3.3 isHit() . . . 23
3.9.3.4 maxDistanceSurpassed() . . . 23
3.9.3.5 setHit() . . . 23
3.9.3.6 updateState() . . . 23
3.10 ScreenEnd Class Reference . . . 23
3.10.1 Detailed Description . . . 24
3.10.2 Constructor & Destructor Documentation . . . 24
3.10.2.1 ScreenEnd() . . . 24
3.10.3 Member Function Documentation . . . 24
3.10.3.1 Run() . . . 24
3.11 ScreenInGame Class Reference . . . 24
3.11.1 Detailed Description . . . 25
3.11.2 Constructor & Destructor Documentation . . . 25
3.11.2.1 ScreenInGame() . . . 25
3.11.3 Member Function Documentation . . . 25
3.11.3.1 Run() . . . 25
3.12 ScreenMenu Class Reference . . . 25
3.12.1 Detailed Description . . . 26
3.12.2 Constructor & Destructor Documentation . . . 26
3.12.2.1 ScreenMenu() . . . 26
3.12.3 Member Function Documentation . . . 26
3.12.3.1 Run() . . . 26
3.13 Tilemap Class Reference . . . 26
3.13.1 Detailed Description . . . 27
3.13.2 Member Function Documentation . . . 27

3.13.2.1 load() . . . 27
3.14 Vehicle Class Reference . . . 27
3.14.1 Detailed Description . . . 28
3.14.2 Constructor & Destructor Documentation . . . 28
3.14.2.1 Vehicle() . . . 28
3.14.3 Member Function Documentation . . . 28
3.14.3.1 getLap() . . . 28
3.14.3.2 setAmmoTexture() . . . 28
3.14.3.3 setTexture() . . . 28
3.14.3.4 shoot() . . . 29
3.14.3.5 updateState() . . . 29
3.15 VehicleInfo Struct Reference . . . 29
3.15.1 Detailed Description . . . 29
3.16 Wall Class Reference . . . 29
3.16.1 Detailed Description . . . 30
3.16.2 Constructor & Destructor Documentation . . . 30
3.16.2.1 Wall() . . . 30
3.16.3 Member Function Documentation . . . 30
3.16.3.1 collide() . . . 30
3.16.3.2 getHeight() . . . 30
3.16.3.3 getWidth() . . . 31
3.17 World Class Reference . . . 31
3.17.1 Detailed Description . . . 31
3.17.2 Member Function Documentation . . . 31
3.17.2.1 addCheckPoints() . . . 31
3.17.2.2 addMap() . . . 32
3.17.2.3 addMines() . . . 32
3.17.2.4 addProjectile() . . . 32
3.17.2.5 addVehicle() . . . 32
3.17.2.6 addVehicleInfo() . . . 32
3.17.2.7 addWall() . . . 32
3.17.2.8 draw() . . . 32
3.17.2.9 getCheckPoints() . . . 33
3.17.2.10 getLapTotal() . . . 33
3.17.2.11 getMap() . . . 33
3.17.2.12 getMines() . . . 33
3.17.2.13 getProjectiles() . . . 33
3.17.2.14 getVehicleInfo() . . . 33
3.17.2.15 getVehicles() . . . 33
3.17.2.16 getWalls() . . . 33
3.17.2.17 setFont() . . . 34
3.17.2.18 setViews() . . . 34
3.17.2.19 updateState() . . . 34

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

cScreen . . . 7
- ScreenEnd . . . 23
- ScreenInGame . . . 24
- ScreenMenu . . . 25

Drawable
- Tilemap . . . 26

GameInfo . . . 8
Map . . . 10
Menu . . . 12
Object . . . 15
- CheckPoint . . . 5
- Mine . . . 12
- Projectile . . . 22
- Vehicle . . . 27
- Wall . . . 29

PlayerInput . . . 19
Transformable
- Tilemap . . . 26

VehicleInfo . . . 29
World . . . 31

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

CheckPoint 5
cScreen 7
GameInfo 8
Map 10
Menu 12
Mine 12
Object 15
PlayerInput 19
Projectile 22
ScreenEnd 23
ScreenInGame 24
ScreenMenu 25
Tilemap 26
Vehicle 27
VehicleInfo 29
Wall 29
World 31

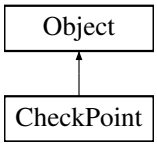
Chapter 3

Class Documentation

3.1 CheckPoint Class Reference

`#include <checkpoint.hpp>`

Inheritance diagram for CheckPoint:



Public Member Functions

- CheckPoint ()
- CheckPoint (Vector2d pos, int num, int width, int angle)
- void collide (Object &o)
- int getWidth () const
- int getHeight () const
- int getNumber () const

Additional Inherited Members

3.1.1 Detailed Description

A class to follow player's progress in game. CheckPoint-object is located in the game area to work as an indicator, whether player has passed a certain point. A integer numbers are set for checkpoints to determint orde in which they have to be visited.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 CheckPoint() [1/2]

`CheckPoint::CheckPoint ( ) [inline]`

Constructor

3.1.2.2 CheckPoint() [2/2]

```
CheckPoint::CheckPoint (
            Vector2d pos,
            int num,
            int width,
            int angle )
```

Constructor

3.1.3 Member Function Documentation

3.1.3.1 collide()

```
void CheckPoint::collide (
            Object & o )  [virtual]
```

Returns the order number of checkpoint. Detects if player has passed the checkpoint.

Implements Object.

3.1.3.2 getHeight()

```
int CheckPoint::getHeight ( ) const  [inline]
```

Returns height of checkpoint.

3.1.3.3 getNumber()

```
int CheckPoint::getNumber ( ) const  [inline]
```

Returns number of checkpoint.

3.1.3.4 getWidth()

```
int CheckPoint::getWidth ( ) const  [inline]
```

Returns width of checkpoint.

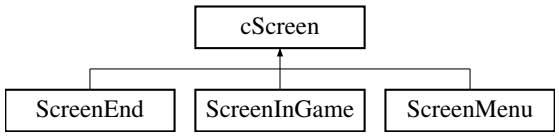
The documentation for this class was generated from the following files:

- src/checkpoint.hpp
- src/checkpoint.cpp

3.2 cScreen Class Reference

`#include <cScreen.hpp>`

Inheritance diagram for cScreen:



Public Member Functions

- cScreen (std::shared_ptr< GameInfo > info)
- virtual int Run (sf::RenderWindow &w)=0
- void set_infoptr (std::shared_ptr< GameInfo > info)
- std::shared_ptr< GameInfo > get_infoptr ()

3.2.1 Detailed Description

A generic class for different scenes or screens inside the game. These include the main menu, the actual game and the end screen.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 cScreen()

```
cScreen::cScreen (
            std::shared_ptr< GameInfo > info )  [inline]
```

Constuctor for the cScreen class.

3.2.3 Member Function Documentation

3.2.3.1 get_infoptr()

```
std::shared_ptr<GameInfo> cScreen::get_infoptr ( )  [inline]
```

Returns the shared information class used by the screen.

3.2.3.2 Run()

```
virtual int cScreen::Run (
            sf::RenderWindow & w )  [pure virtual]
```

A function to run the scene. The actual gameplay or a menu is completely contained inside this function.

Implemented in ScreenMenu, ScreenEnd, and ScreenInGame.

3.2.3.3 set_infoptr()

```
void cScreen::set_infoptr (
            std::shared_ptr< GameInfo > info )  [inline]
```

Sets the shared information class to be used between different screens for the screen.

The documentation for this class was generated from the following file:

- src/cScreen.hpp

3.3 GameInfo Class Reference

```
#include <gameinfo.hpp>
```

Public Member Functions

- GameInfo ()
- void set_map_path (std::string path)
- std::string get_map_path ()
- void set_tilemap_path (std::string tpath)
- std::string get_tilemap_path ()
- void set_lap_total (int number)
- int get_lap_total ()
- std::shared_ptr< PlayerInput > getPlayerInput (int p)
- void setPlayerInput (PlayerInput const &p, int player)

3.3.1 Detailed Description

Class for storing user-defined game settings that is useful to all game screens.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 GameInfo()

```
GameInfo::GameInfo ( )  [inline]
```

Constructor

3.3.3 Member Function Documentation

3.3.3.1 get_lap_total()

```
int GameInfo::get_lap_total ( )  [inline]
```

Returns the amount of laps in a race.

3.3.3.2 get_map_path()

```
std::string GameInfo::get_map_path ( )  [inline]
```

Returns the current path for mapfile for map generation.

3.3.3.3 get_tilemap_path()

```
std::string GameInfo::get_tilemap_path ( )  [inline]
```

Returns the current path for tilemap-file for map generation.

3.3.3.4 getPlayerInput()

```
std::shared_ptr<PlayerInput> GameInfo::getPlayerInput (
            int p )  [inline]
```

Returns PlayerInput-object pointer for game controls of player

3.3.3.5 set_lap_total()

```
void GameInfo::set_lap_total (
            int number )  [inline]
```

Setting the amount of laps in a race.

3.3.3.6 set_map_path()

```
void GameInfo::set_map_path (
            std::string path )  [inline]
```

Setting path of mapfile for map generation.

3.3.3.7 set_tilemap_path()

```
void GameInfo::set_tilemap_path (
            std::string tpath )  [inline]
```

Setting path for tilemap-file for map generation.

3.3.3.8 setPlayerInput()

```
void GameInfo::setPlayerInput (
            PlayerInput const & p,
            int player )  [inline]
```

Sets controls for player through PlayerInput-object.

The documentation for this class was generated from the following file:

- src/gameinfo.hpp

3.4 Map Class Reference

`#include <map.hpp>`

Public Member Functions

- Map (const char *filename, std::shared_ptr< World >)
- void setTilesetPath (std::string const &s)
- bool loadTileMap ()
- void createWalls (std::shared_ptr< World > w)
- Tilemap & getTilemap ()
- void draw (Window &w)
- int getHeight ()
- int getWidth ()
- sf::Vector2u getTileSize ()
- std::vector< std::shared_ptr< CheckPoint > > getCheckPoints ()
- std::vector< std::shared_ptr< Mine > > getMines ()

3.4.1 Detailed Description

Map-class reads game field from a text file in its constructor. Text file consists of characters representing tiles. The text file is assumed to have equally many characters in every row and it ends in newline-character

3.4.2 Constructor & Destructor Documentation

3.4.2.1 Map()

```
Map::Map (
            const char * filename,
            std::shared_ptr< World > w )
```

Constructor for map. Loads the map from a map file.

3.4.3 Member Function Documentation

3.4.3.1 createWalls()

```
void Map::createWalls (
            std::shared_ptr< World > w )
```

Creates walls according to map structure.

3.4.3.2 draw()

```
void Map::draw (
            Window & w )
```

Draws the tilemap.

3.4.3.3 getCheckPoints()

```
std::vector<std::shared_ptr<CheckPoint> > Map::getCheckPoints ( )  [inline]
```

Returns the checkpoints as a vector.

3.4.3.4 getHeight()

```
int Map::getHeight ( )  [inline]
```

Returns the height of the map.

3.4.3.5 getMines()

```
std::vector<std::shared_ptr<Mine> > Map::getMines ( )  [inline]
```

Returns the mines of the map as a list

3.4.3.6 getTilemap()

```
Tilemap& Map::getTilemap ( )  [inline]
```

Returns the tilemap structure.

3.4.3.7 getTileSize()

```
sf::Vector2u Map::getTileSize ( )  [inline]
```

Returns the tile size as a vector.

3.4.3.8 getWidth()

```
int Map::getWidth ( )  [inline]
```

Returns the idth of the map.

3.4.3.9 loadTileMap()

```
bool Map::loadTileMap ( )
```

Loads the tilemap according to the map structure and tileset.

3.4.3.10 setTilesetPath()

```
void Map::setTilesetPath (
            std::string const & s )  [inline]
```

Sets the path for the tileset path to be used.

The documentation for this class was generated from the following files:

- src/map.hpp
- src/map.cpp

3.5 Menu Struct Reference

Public Attributes

- int **main_op** =0
- int **map_choice** =0
- int **player_choice** =0
- int **key** =0
- int **chosen_map** = 0
- bool **in_main** =true
- bool **in_map** =false
- bool **in_keys** =false
- bool **in_p** = false

3.5.1 Detailed Description

A structure for holding the menu state in.

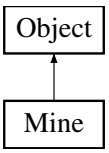
The documentation for this struct was generated from the following file:

- src/screen_menu.cpp

3.6 Mine Class Reference

```
#include <mine.hpp>
```

Inheritance diagram for Mine:



Public Member Functions

- Mine (Vector2d, int)
- virtual void collide (Object &o)
- virtual void setTexture (sf::Texture &)
- void updateState ()
- bool isVisible ()
- void setInvisible ()
- void detonate ()
- void draw (Window &w)
- sf::Clock & getClock ()

Additional Inherited Members

3.6.1 Detailed Description

A class for the sole obstacle in the game.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 Mine()

```
Mine::Mine (
            Vector2d pos,
            int rad )
```

Constructor

3.6.3 Member Function Documentation

3.6.3.1 collide()

```
void Mine::collide (
            Object & o )  [virtual]
```

Collides the mine with another object.

Implements Object.

3.6.3.2 detonate()

```
void Mine::detonate ( )
```

Detonates the mine when it comes to contact with a vehicle.

3.6.3.3 draw()

```
void Mine::draw (
            Window & w )
```

Draws the mine on the window.

3.6.3.4 getClock()

```
sf::Clock& Mine::getClock ( )
```

Returns the clock that controls the cooldown of the mine.

3.6.3.5 isVisible()

```
bool Mine::isVisible ( )
```

Checks whether the mine is inactive or not.

3.6.3.6 setInvisible()

```
void Mine::setInvisible ( )
```

Sets the mine inactive.

3.6.3.7 setTexture()

```
void Mine::setTexture (
            sf::Texture & tex )  [virtual]
```

Sets the texture for the mine.

Reimplemented from Object.

3.6.3.8 updateState()

```
void Mine::updateState ( )
```

Updates the mine's state, including its position.

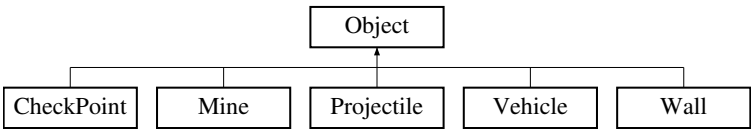
The documentation for this class was generated from the following files:

- src/mine.hpp
- src/mine.cpp

3.7 Object Class Reference

`#include <object.hpp>`

Inheritance diagram for Object:



Public Member Functions

- const Vector2d & getPosition () const
- void setPosition (Vector2d const &p)
- const Vector2d & getSpeed () const
- void setSpeed (Vector2d const &s)
- virtual void setTexture (sf::Texture &)
- double getMass () const
- void setMass (double const &m)
- double getAngle () const
- void setAngle (double const &a)
- sf::Sprite getSprite () const
- void draw (Window &w)
- double getRadius () const

Protected Member Functions

- bool collides (Object const &o) const
- bool collides (Wall const &w) const
- bool collides (CheckPoint const &cp) const
- virtual void **collide** (Object &o)=0

Protected Attributes

- Vector2d position
- Vector2d speed
- double angle
- double mass
- double radius
- sf::Sprite sprite
- sf::Texture texture

3.7.1 Detailed Description

Generic class for game objects, such players, checkpoints and projectiles.

3.7.2 Member Function Documentation

3.7.2.1 collides() [1/3]

```
bool Object::collides (
            Object const & o ) const  [protected]
```

Checks if an object collides with another object.

3.7.2.2 collides() [2/3]

```
bool Object::collides (
            Wall const & w ) const  [protected]
```

Checks if an object collides with a wall.

3.7.2.3 collides() [3/3]

```
bool Object::collides (
            CheckPoint const & cp ) const  [protected]
```

checks if an object collides with a map checkpoint.

3.7.2.4 draw()

```
void Object::draw (
            Window & w )
```

Draws an object on a window.

3.7.2.5 getAngle()

```
double Object::getAngle ( ) const
```

Returns the angle of an object in degrees.

3.7.2.6 getMass()

```
double Object::getMass ( ) const
```

Returns the mass of an object.

3.7.2.7 getPosition()

```
const Vector2d & Object::getPosition ( ) const
```

Returns the objects position in pixels as a 2d vector.

3.7.2.8 getRadius()

```
double Object::getRadius ( ) const
```

Returns the radius of an object.

3.7.2.9 getSpeed()

```
const Vector2d & Object::getSpeed ( ) const
```

Returns the speed of an object.

3.7.2.10 getSprite()

```
sf::Sprite Object::getSprite ( ) const
```

Returns the sprite of an object.

3.7.2.11 setAngle()

```
void Object::setAngle (
            double const & a )
```

Sets the angle of an object in degrees.

3.7.2.12 setMass()

```
void Object::setMass (
            double const & m )
```

Sets the mass of an object.

3.7.2.13 setPosition()

```
void Object::setPosition (
            Vector2d const & p )
```

Sets the position of an object.

3.7.2.14 setSpeed()

```
void Object::setSpeed (
            Vector2d const & s )
```

Sets the speed of an object.

3.7.2.15 setTexture()

```
virtual void Object::setTexture (
            sf::Texture &  )  [inline], [virtual]
```

virtual function for the setting of a texture.

Reimplemented in Mine.

3.7.3 Member Data Documentation

3.7.3.1 angle

```
double Object::angle  [protected]
```

Angle of the object in degrees.

3.7.3.2 mass

```
double Object::mass  [protected]
```

Mass of the object.

3.7.3.3 position

```
Vector2d Object::position  [protected]
```

Position of the object as a vector.

3.7.3.4 radius

```
double Object::radius  [protected]
```

Radius of the object.

3.7.3.5 speed

```
Vector2d Object::speed  [protected]
```

Speed of the object as a vector.

3.7.3.6 sprite

```
sf::Sprite Object::sprite  [protected]
```

Sprite of the object.

3.7.3.7 texture

```
sf::Texture Object::texture  [protected]
```

Texture of the object.

The documentation for this class was generated from the following files:

- src/object.hpp
- src/object.cpp

3.8 PlayerInput Class Reference

```
#include <playerinput.hpp>
```

Public Member Functions

- sf::Keyboard::Key getKeyUp () const
- void setKeyUp (sf::Keyboard::Key k)
- sf::Keyboard::Key getKeyDown () const
- void setKeyDown (sf::Keyboard::Key k)
- sf::Keyboard::Key getKeyRight () const
- void setKeyRight (sf::Keyboard::Key k)
- sf::Keyboard::Key getKeyLeft () const
- void setKeyLeft (sf::Keyboard::Key k)
- sf::Keyboard::Key getKeyShoot () const
- void setKeyShoot (sf::Keyboard::Key k)
- bool Up ()
- bool Down ()
- bool Left ()
- bool Right ()
- bool Shoot ()

3.8.1 Detailed Description

A class for taking input from player.

3.8.2 Member Function Documentation

3.8.2.1 Down()

```
bool PlayerInput::Down ( )
```

Returns whether player is pressing his designated down key.

3.8.2.2 getKeyDown()

```
sf::Keyboard::Key PlayerInput::getKeyDown ( ) const
```

Returns player down key value.

3.8.2.3 getKeyLeft()

```
sf::Keyboard::Key PlayerInput::getKeyLeft ( ) const
```

Returns player left key value.

3.8.2.4 getKeyRight()

```
sf::Keyboard::Key PlayerInput::getKeyRight ( ) const
```

Returns player right key value.

3.8.2.5 getKeyShoot()

```
sf::Keyboard::Key PlayerInput::getKeyShoot ( ) const
```

Returns player shoot key value.

3.8.2.6 getKeyUp()

```
sf::Keyboard::Key PlayerInput::getKeyUp ( ) const
```

Returns player up key value.

3.8.2.7 Left()

```
bool PlayerInput::Left ( )
```

Returns whether player is pressing his designated left key.

3.8.2.8 Right()

```
bool PlayerInput::Right ( )
```

Returns whether player is pressing his designated right key.

3.8.2.9 setKeyDown()

```
void PlayerInput::setKeyDown (
            sf::Keyboard::Key k )
```

Sets player down key value.

3.8.2.10 setKeyLeft()

```
void PlayerInput::setKeyLeft (
            sf::Keyboard::Key k )
```

Sets player left key value.

3.8.2.11 setKeyRight()

```
void PlayerInput::setKeyRight (
            sf::Keyboard::Key k )
```

Sets player right key value.

3.8.2.12 setKeyShoot()

```
void PlayerInput::setKeyShoot (
            sf::Keyboard::Key k )
```

Sets player shoot key value.

3.8.2.13 setKeyUp()

```
void PlayerInput::setKeyUp (
            sf::Keyboard::Key k )
```

Sets player up key value.

3.8.2.14 Shoot()

```
bool PlayerInput::Shoot ( )
```

Returns whether player is pressing his designated shoot key.

3.8.2.15 Up()

```
bool PlayerInput::Up ( )
```

Returns whether player is pressing his designated up key.

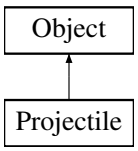
The documentation for this class was generated from the following files:

- src/playerinput.hpp
- src/playerinput.cpp

3.9 Projectile Class Reference

`#include <projectile.hpp>`

Inheritance diagram for Projectile:



Public Member Functions

- Projectile (Vector2d pos, Vector2d sp, double m, sf::Texture &tex, std::shared_ptr< World > w)
- void updateState (double dt)
- void collide (Wall &w)
- virtual void collide (Object &o)
- bool isHit ()
- void setHit ()
- bool maxDistanceSurpassed ()

Additional Inherited Members

3.9.1 Detailed Description

A class for the projectiles.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 Projectile()

```
Projectile::Projectile (
            Vector2d pos,
            Vector2d sp,
            double m,
            sf::Texture & tex,
            std::shared_ptr< World > w )
```

Constructor

3.9.3 Member Function Documentation

3.9.3.1 collide() [1/2]

```
void Projectile::collide (
            Wall & w )
```

Collides the projectile with a wall.

3.9.3.2 collide() [2/2]

```
void Projectile::collide (
            Object & o )  [virtual]
```

Collides the projectile with an object.

Implements Object.

3.9.3.3 isHit()

```
bool Projectile::isHit ( )  [inline]
```

Returns whether the object is hit or not.

3.9.3.4 maxDistanceSurpassed()

```
bool Projectile::maxDistanceSurpassed ( )
```

Checks whether the projectile has traveled its maximum distance.

3.9.3.5 setHit()

```
void Projectile::setHit ( )  [inline]
```

Sets the projectile as hit.

3.9.3.6 updateState()

```
void Projectile::updateState (
            double dt )
```

Updates the state of the projectile, meaning both its position and sprite.

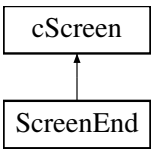
The documentation for this class was generated from the following files:

- src/projectile.hpp
- src/projectile.cpp

3.10 ScreenEnd Class Reference

```
#include <screen_end.hpp>
```

Inheritance diagram for ScreenEnd:



Public Member Functions

- ScreenEnd (std::shared_ptr< GameInfo > info)
- int Run (sf::RenderWindow &w)

3.10.1 Detailed Description

End screen.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 ScreenEnd()

```
ScreenEnd::ScreenEnd (
            std::shared_ptr< GameInfo > info )  [inline]
```

Constructor for end screen.

3.10.3 Member Function Documentation

3.10.3.1 Run()

```
int ScreenEnd::Run (
            sf::RenderWindow & w )  [virtual]
```

The actual end screen functionality.

Implements cScreen.

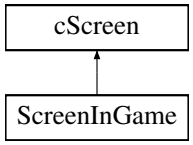
The documentation for this class was generated from the following files:

- src/screen_end.hpp
- src/screen_end.cpp

3.11 ScreenInGame Class Reference

```
#include <screen_ingame.hpp>
```

Inheritance diagram for ScreenInGame:



Public Member Functions

- ScreenInGame (std::shared_ptr< GameInfo > info)
- int Run (sf::RenderWindow &)

3.11.1 Detailed Description

Screen that contains the actual gameplay. Shares information with other sceens by using the GameInfo class.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 ScreenInGame()

```
ScreenInGame::ScreenInGame (
            std::shared_ptr< GameInfo > info )  [inline]
```

Constructor for the gameplay screen.

3.11.3 Member Function Documentation

3.11.3.1 Run()

```
int ScreenInGame::Run (
            sf::RenderWindow & w )  [virtual]
```

A function to run the scene. The actual gameplay or a menu is completely contained inside this function.

Implements cScreen.

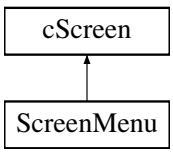
The documentation for this class was generated from the following files:

- src/screen_ingame.hpp
- src/screen_ingame.cpp

3.12 ScreenMenu Class Reference

```
#include <screen_menu.hpp>
```

Inheritance diagram for ScreenMenu:



Public Member Functions

- ScreenMenu (std::shared_ptr< GameInfo > info)
- int Run (sf::RenderWindow &w)

3.12.1 Detailed Description

Class dor the menu screen.

3.12.2 Constructor & Destructor Documentation

3.12.2.1 ScreenMenu()

```
ScreenMenu::ScreenMenu (
            std::shared_ptr< GameInfo > info )  [inline]
```

Constructor for the menu screen.

3.12.3 Member Function Documentation

3.12.3.1 Run()

```
int ScreenMenu::Run (
            sf::RenderWindow & w )  [virtual]
```

The actual menu screen functionality.

Implements cScreen.

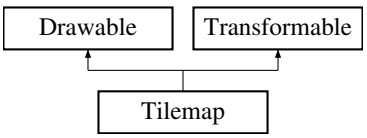
The documentation for this class was generated from the following files:

- src/screen_menu.hpp
- src/screen_menu.cpp

3.13 Tilemap Class Reference

```
#include <tilemap.hpp>
```

Inheritance diagram for Tilemap:



Public Member Functions

- bool load (std::string const &tileSetPath, sf::Vector2u tilesize, Matrix map, int width, int height)

3.13.1 Detailed Description

A class for the drawing of the game world map. Tilemap contains the tiles that form the game map. This class is a slightly modified version of an example written by Laurent Gomila, licensed under the zlib/png license.

3.13.2 Member Function Documentation

3.13.2.1 load()

```
bool Tilemap::load (
            std::string const & tileSetPath,
            sf::Vector2u tilesize,
            Matrix map,
            int width,
            int height )
```

Creates the tilemap from a repserentation of the map as well as a tileset.

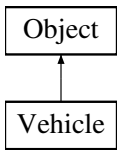
The documentation for this class was generated from the following files:

- src/tilemap.hpp
- src/tilemap.cpp

3.14 Vehicle Class Reference

```
#include <vehicle.hpp>
```

Inheritance diagram for Vehicle:



Public Member Functions

- Vehicle (Vector2d pos, double orientationAngle, double maxSp, double maxF, double m, std::shared_ptr< PlayerInput > p1, std::shared_ptr< World > w)
- void updateState (double dt)
- virtual void setTexture (std::string path)
- void setAmmoTexture (std::string path)
- void shoot (std::shared_ptr< World > world)
- int getLap () const

Additional Inherited Members

3.14.1 Detailed Description

A class for containing the vehicles in the game. A vehicle is derived from the object class, and therefore has the physical properties that an object has.

3.14.2 Constructor & Destructor Documentation

3.14.2.1 Vehicle()

```
Vehicle::Vehicle (
            Vector2d pos,
            double orientationAngle,
            double maxSp,
            double maxF,
            double m,
            std::shared_ptr< PlayerInput > p1,
            std::shared_ptr< World > w )
```

Constructor

3.14.3 Member Function Documentation

3.14.3.1 getLap()

```
int Vehicle::getLap ( ) const  [inline]
```

Returns the lap the vehicle is currently on.

3.14.3.2 setAmmoTexture()

```
void Vehicle::setAmmoTexture (
            std::string path )
```

Sets a texture for the ammo type the vehicle uses.

3.14.3.3 setTexture()

```
void Vehicle::setTexture (
            std::string path )  [virtual]
```

Sets the texture for the vehicle.

3.14.3.4 shoot()

```
void Vehicle::shoot (
            std::shared_ptr< World > world )
```

This function handles the shooting of a projectile.

3.14.3.5 updateState()

```
void Vehicle::updateState (
            double dt )
```

Updates the state of the vehicle. This includes both the physics as well as the input for the vehicle.

The documentation for this class was generated from the following files:

- src/vehicle.hpp
- src/vehicle.cpp

3.15 VehicleInfo Struct Reference

```
#include <vehicle.hpp>
```

Public Attributes

- Vector2d **position1**
- Vector2d **position2**
- double **orientation**
- double **maxSp**
- double **maxF**
- double **mass**

3.15.1 Detailed Description

A structure for setting the initial configurations for vehicles in a map.

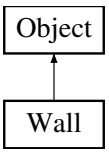
The documentation for this struct was generated from the following file:

- src/vehicle.hpp

3.16 Wall Class Reference

```
#include <wall.hpp>
```

Inheritance diagram for Wall:



Public Member Functions

- Wall (Vector2d pos, int w, int h, double orientationAngle=0)
- void collide (Object &o)
- const double & getWidth () const
- const double & getHeight () const

Additional Inherited Members

3.16.1 Detailed Description

A class for the walls in our game world.

3.16.2 Constructor & Destructor Documentation

3.16.2.1 Wall()

```
Wall::Wall (
            Vector2d pos,
            int w,
            int h,
            double orientationAngle = 0 )  [inline]
```

Constructor

3.16.3 Member Function Documentation

3.16.3.1 collide()

```
void Wall::collide (
            Object & o )  [virtual]
```

A function for colliding with objects.

Implements Object.

3.16.3.2 getHeight()

```
const double & Wall::getHeight ( ) const
```

Returns the height of the wall.

3.16.3.3 getWidth()

```
const double & Wall::getWidth ( ) const
```

Returns the width of the wall

The documentation for this class was generated from the following files:

- src/wall.hpp
- src/wall.cpp

3.17 World Class Reference

```
#include <world.hpp>
```

Public Member Functions

- void addVehicle (VehiclePtr v)
- void addProjectile (ProjectilePtr p)
- void addWall (WallPtr w)
- void addCheckPoints (std::vector< CpPtr > c)
- void addMines (std::vector< MinePtr > m)
- void addMap (MapPtr m)
- void addVehicleInfo (std::shared_ptr< VehicleInfo >)
- std::vector< VehiclePtr > getVehicles () const
- std::vector< ProjectilePtr > getProjectiles () const
- MapPtr getMap () const
- std::shared_ptr< VehicleInfo > getVehicleInfo () const
- std::vector< WallPtr > getWalls () const
- std::vector< CpPtr > getCheckPoints () const
- std::vector< MinePtr > getMines () const
- int getLapTotal () const
- std::vector< sf::View > setViews (Window &w) const
- void setFont (std::shared_ptr< sf::Font > f)
- void updateState (double dt)
- void draw (Window &w)

3.17.1 Detailed Description

A class that contains the game world and everything in it. Is responsible for updating the game logic as well as drawing the game world.

3.17.2 Member Function Documentation

3.17.2.1 addCheckPoints()

```
void World::addCheckPoints (
            std::vector< CpPtr > c )
```

Adds the map checkpoints to world.

3.17.2.2 addMap()

```
void World::addMap (
            MapPtr m )
```

Adds the game map to world.

3.17.2.3 addMines()

```
void World::addMines (
            std::vector< MinePtr > m )
```

Adds mines to world.

3.17.2.4 addProjectile()

```
void World::addProjectile (
            ProjectilePtr p )
```

Adds a projectile smart pointer to world.

3.17.2.5 addVehicle()

```
void World::addVehicle (
            VehiclePtr v )
```

Adds a vehicle smart pointer to world.

3.17.2.6 addVehicleInfo()

```
void World::addVehicleInfo (
            std::shared_ptr< VehicleInfo >  )
```

Reads vehicle info from maps

3.17.2.7 addWall()

```
void World::addWall (
            WallPtr w )
```

Adds a wall smart pointer to world.

3.17.2.8 draw()

```
void World::draw (
            Window & w )
```

Draws the game world to a window.

3.17.2.9 getCheckPoints()

```
std::vector< std::shared_ptr< CheckPoint > > World::getCheckPoints ( ) const
```

gets the map checkpoints.

3.17.2.10 getLapTotal()

```
int World::getLapTotal ( ) const  [inline]
```

Returns the total amount of laps in the race.

3.17.2.11 getMap()

```
MapPtr World::getMap ( ) const
```

Returns the game map

3.17.2.12 getMines()

```
std::vector< std::shared_ptr< Mine > > World::getMines ( ) const
```

Gets a list of mines from world.

3.17.2.13 getProjectiles()

```
std::vector< ProjectilePtr > World::getProjectiles ( ) const
```

A method to get the projectiles inside the world

3.17.2.14 getVehicleInfo()

```
std::shared_ptr< VehicleInfo > World::getVehicleInfo ( ) const
```

A Method to fetch vehicle info from world.

3.17.2.15 getVehicles()

```
std::vector< VehiclePtr > World::getVehicles ( ) const
```

A method to get the vehicles inside the world

3.17.2.16 getWalls()

```
std::vector< WallPtr > World::getWalls ( ) const
```

gets the walls from world.

3.17.2.17 setFont()

```
void World::setFont (
            std::shared_ptr< sf::Font > f )
```

Sets a font for world.

3.17.2.18 setViews()

```
std::vector< sf::View > World::setViews (
            Window & w ) const
```

Sets the split-screen view necessary for 2-player play and returns a list of the views.

3.17.2.19 updateState()

```
void World::updateState (
            double dt )
```

Updates the game state. This includes both the game physics as well as player input.

The documentation for this class was generated from the following files:

- src/world.hpp
- src/world.cpp