Hills Racing

Project documentation

ELEC-A7150 C++ programming

Hill-racing-1
Paavo Hietala
Veera Itälinna
Oskari Lehtonen
Markus Toivonen

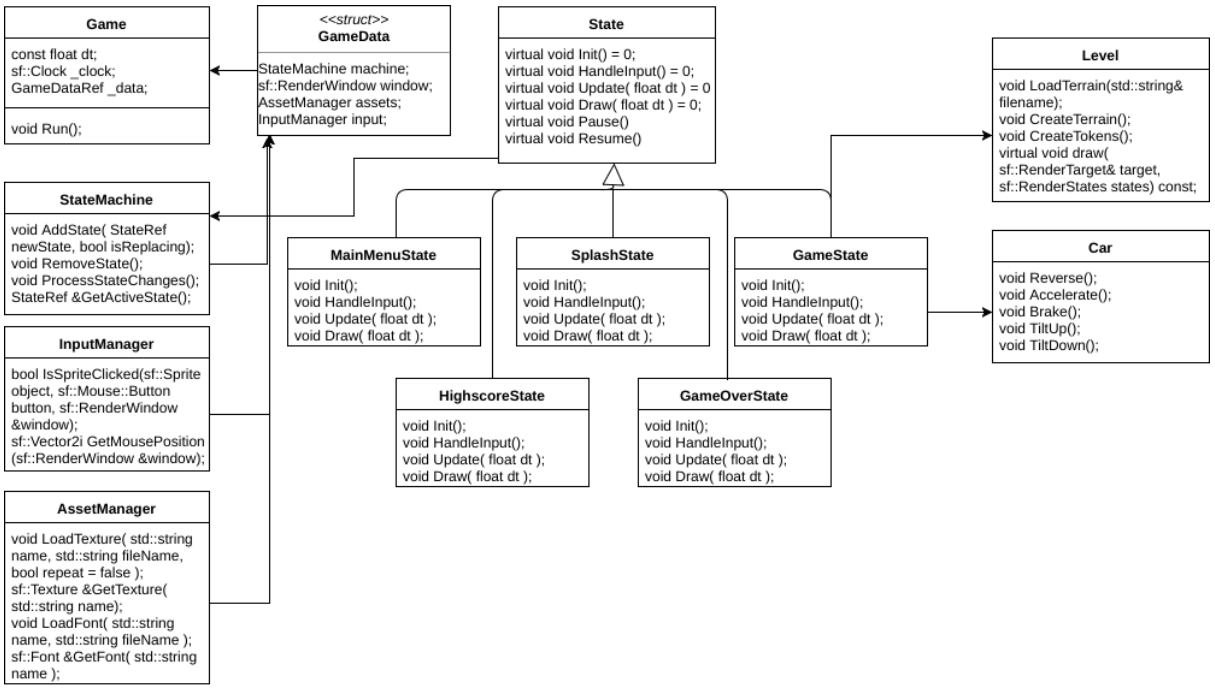
1. Overview

As our course project we created our own version of the classic Hill Climb Racing game. We used Box2D physics engine and SFML media library, which are open source C++ libraries.

The goal of the game is to drive a motorized vehicle through a hilly 2D track as fast as possible while collecting tokens along the way. The game ends when the player reaches the goal, gets stuck upside down or falls off screen. When the player reaches the finish line, a score is calculated based on the elapsed time as well as the number of tokens collected. Reaching the finish line or getting stuck upside down will present the game over screen which shows the score for the run and instructions for restarting or navigating to the main menu. The top 3 scores for each track are saved in the “High Scores” table. In addition to the vehicle and the track, the user interface of the game shows the amount of time elapsed, current score and car speed.

2. Software structure

The class structure with the most important methods and fields is illustrated in the UML diagram below.



The Game class runs the main loop of the game. The game runs at a fixed time delta value, while allowing the game to render the graphics at different frame rates using interpolation. The Game instance holds a shared pointer to a GameData struct, which contains the game window and instances of StateMachine, AssetManager and InputManager. The shared pointer allows this data to be safely used in multiple parts of the game without needing to worry about releasing the memory.

The AssetManager class has functions for loading images and fonts needed in the game, as well as fetching the loaded resources to use. The resources are stored in a map container, where they can be accessed with a keyword. The InputManager provides a simple interface for checking if a sprite is clicked. The StateMachine class controls the state changes, such as transitioning from main menu to gameplay. The different states are stored in a stack, where the state that's currently active is on top. When a new state is added elsewhere in the program, its ownership is moved to StateMachine using a unique pointer, so each unused state is safely removed when the pointer is popped out of stack.

The State class is a base class for the different game states. The Init, HandleInput, Update and Draw functions are pure virtual functions that are implemented in the classes inheriting from State.

When a level is chosen in Main Menu state, a Game state is added on top of the states stack with the appropriate level file path as a parameter. The level text files define the terrain shape by storing the y-coordinates of the drawing points. Token positions are also defined in these files: on each line of the file, the first number represents the terrain coordinate and the possible second number is the token coordinate. The level file path is passed on to the constructor of the Level class, which creates the track and adds tokens according to the coordinates. The ground is created as both physical Box2D ground and a drawable SFML vertex array.

The Game state also creates a Car class instance, handles keyboard controls and view movement, keeps track of car position, angle and player score and draws everything in place. When the player reaches the finish line, the score is saved in a text file.

The Car class defines the physical properties of the vehicle and creates drawable sprites for the chassis and wheels. The car physics is implemented with Box2D bodies and wheel joints. The class also contains functions for car movement according to user controls.

Rule of three is followed in the GameState class, where Level and Car objects are dynamically allocated. The class has a destructor which deletes the level and car. Copying of GameState is prevented. In other parts of the program dynamic memory is handled with smart pointers, so the classes release the memory automatically.

3. Instructions for building and using the software

How to compile and run the program (Linux)

Running the program requires SFML to be installed on the computer. Instructions for installation can be found at https://www.sfml-dev.org/tutorials/2.4/start-linux.php.

In the `hill-racing-1/src` folder, run the terminal command '`make`' to compile the program. To start the app, run the command '`./hills-racing`'.

How to play

In the main menu, you can choose a level or view high scores by clicking the appropriate button on screen. The three levels have increasing difficulty.

The game starts immediately after selecting a level. The car can be controlled with the following controls:

- Up/W: Accelerate
- Down/S: Decelerate
- Left/A: Tilt up
- Right/D: Tilt down
- Restart: R
- Main menu: Esc

You receive points based on the finishing time and the number of star-shaped tokens collected. The top three scores for each level are saved.

If the car gets stuck upside down or falls off screen, you lose the game. You can return to the main menu during the game or from the Game Over screen by pressing Esc on the keyboard. You can restart the current map by pressing R during gameplay or in the Game Over screen.

4. Testing

Graphics drawing and car movement were tested simply by visual inspection. Some numeric values, such as sprite coordinates, were tested by printing them on console.

The physics properties and parameters of the car were initially tested using the Box2D Testbed, where we created our own test class based on the included “Car”-test.

Valgrind was used to check for memory leaks.

5. Work log

Division of work

Markus.

- HighscoreState: Traversing in and out of the state and showing the top 3 points for each map
- GameOverState: once finishline is reached, it jumps to the gameoverstate and also saves and shows your points for each map. Also saves your points if you fall into a pit / your car turns upside down
- Interface creation for GameState (creating the stopwatch, creating the point counter, and counting the velocity of the car)

Veera

- Terrain and car drawing
- Moving the view
- Initial controls for car
- Loading levels from text files and choosing the level from main menu
- Collecting tokens
- Testing with Box2D testbed

Paavo

- Makefile
- Advanced car controls & car property tuning
- Game over if car stops upside down
- Navigation between states and R for restart
- Textures and other visuals
- Level-specific visuals

Oskari

- Game engine
- Sprites for the vehicle
- Box2d physics implementation
- Box2D Debug Draw Mode
- All around hassling (free ride mode in testmain.cpp)
- Physics Testing
- Sounds

Weekly log

Week 46

- Markus: Project plan done together with the team
- Veera: Project plan, studying resources 5h
- Paavo: Project plan, familiarizing with the addon libraries 4h + wasting time on failing ubuntu installations on two machines 8+h
- Oskari: Project plan, installing libraries and trying to make them work without much success, 5h

Week 47

- Markus: -
- Veera: -
- Paavo: Busy with other projects 0h
- Oskari: Game engine, searching learning material, 7h

Week 48 (mid-term meeting week)

- Markus: Trying to get SFML and Box2D to work on two different work stations, hours: 8h
- Veera: Installing libraries, Box2D Testbed setup, first versions of terrain creation (first flat ground, then randomized hills), moving the view with keyboard 12h
- Paavo: Attempts on creating the car, done makefile, box2d testing 8h
- Oskari: Attempts on creating the car. Learning Box2D and trying to link it with SFML, 12h

Week 49

- Markus: overviewing the code done so far by other members of the team, learning division of classes and functions
- Veera: Researching better ways for terrain creation 4h
- Paavo: Figuring missing parts in theory and pseudo-coding inside my head, soaking in SFML and Box2D tutorials 8h
- Oskari: Learning to do some cool physics with Box2D and drawing sprites on the b2bodies 15h

Week 50

- Markus: Completed all my parts mentioned in division of work. Hours ~ 30
- Veera: Car drawing and movement; loading levels from files; finish line and tokens; selecting the level from main menu; visual fixes, refactoring and code cleanup; designing levels; documentation ~30h
- Paavo: New textures (level-specific), GameOverState triggers, ingame navigation, car and other variable tuning, code reformatting, highscore tweaking, controls adjustments, added grass on dirt, bugfixes & documentation 26h
- Oskari: Completed all the rest mentioned in division of work ~30h