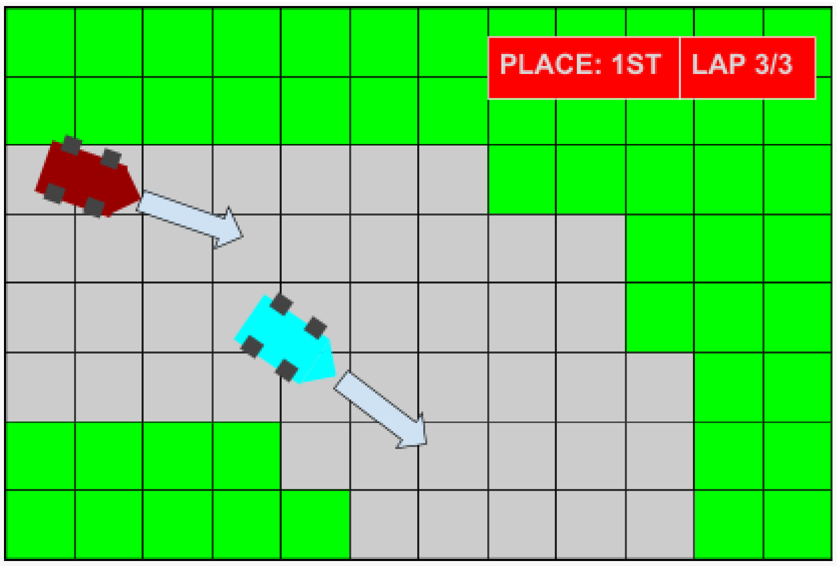
Sampsa Hyvämäki
Arto Lehisto (PM)
Tuomas Poutanen
Mikko Murhu

Micro Machines - Project Plan

Micro Machines is a driving game with versatile vehicle and map options. During the project we'll implement a few different vehicle models and maps. The game will have local multiplayer using splitscreen and also some sort of artificial intelligence to be competed against. The user interface will include a menu, where setup of game options is managed. From the menu the user can launch the game or exit the game. In game, a self-created physics engine for objects' movement will be provided. For each vehicle we create a unique firing mechanism and controllability. Vehicles are map-dependent like in the original game. Maps consist of main course with a couple of checkpoints and both stable and moving obstacles. The game ends when all but one player reach three rounds. After a game, the user will return to the menu. A record of best times for each individual map will be kept, and shown to user after game.



Mock-up of the game screen

Scope of the project

Physics:
Each object's position is defined by a 2-dimensional position-vector. If the object needs to move, it needs to have a vector for speed, and if the object's speed changes through time, it needs to have a vector for acceleration or force. The object might also need a vector to keep track of the object's orientation.
The movement of objects is based on forward Euler integration. At each time step we apply maximum acceleration to the direction of the orientation vector. The orientation vector can be altered by either user input or AI's algorithms. After that we update the object's speed by adding together the acceleration and speed vectors. If the speed would exceed the defined maximum speed of the object, it is truncated to the maximum speed instead. We then update the object's position by adding together the speed and position vectors. With a small time step this produces sufficiently smooth motion for the purpose of this project.

Objects can interact with each other, for example vehicles may bump into each other, or get hit by projectiles. These interactions require collision detection. When objects seem to touch each other on the screen, a collision should be detected and both objects act to the collision as necessary.

Game:
World-class contains all game-relevant information necessary (e.g. “game world”, players, scoreboard, game state (menu/in-game), map etc.), and handles updating of the game. Class calls for all sub-instances to update their state and renders the game situation for graphical user-interface.

Map:
Plan is to make tile-based environment, where “platform” is made of squares. Each square has a property (forest, mud, track…) and when put next to other squares form the game area and, for example, a track for player to follow and hazards to avoid. Some tiles have property to cause collision with player’s car or other effects. It also includes start/finish and a couple of checkpoints to ensure the route has been followed. For AI, there are either certain objects in the map for them to follow (e.g. ports), or the map contains a vector flux field, which determines the direction for AI to steer and accelerate in a specific point in the map.

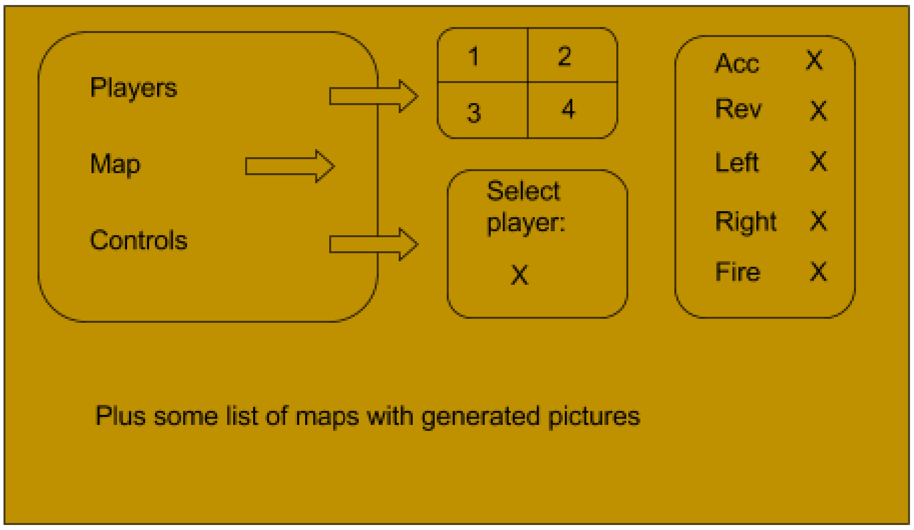
Vehicle:
The vehicle object is able to move forwards and backwards. Acceleration takes a marginal time after which the vehicle’s speed will cap to a certain limit. The vehicle turns left and right with a constant angular velocity according to input. Turning can occur also when stationary. The vehicle can not move through obstacles nor other vehicles. When a collision between vehicles or a vehicle and an obstacle happens, vehicles bump out from collision center according to vehicle's’ momentum.

SFML:
We’ll use SFML as the multimedia library for this project. SFML provides us with ready-made tools for drawing our game world, as well as handling input from the user. SFML also makes it possible for us to implement sounds and music in our game by using SFML’s ready-made libraries.

Graphics:
The game has a few different types of entities that we will have to draw. During gameplay, all of the relevant game objects, such as vehicles, obstacles and the map itself will be drawn on the player’s view of the game world. This view follows the player’s vehicle. In The game world is going to be tile-based, that is, the different terrains consist of repeated tiles, to make it possible to make maps of arbitrary size. The menus in the game consist of menu objects, such as pictures and text. We’ll investigate the possibility of implementing animation through changing sprites for e.g. water wake effects, burning rubber, etc.

Input:
The program takes user input for four directions of movement and firing either from keyboard or joystick.

Sampsa Hyvämäki
Arto Lehisto (PM)
Tuomas Poutanen
Mikko Murhu

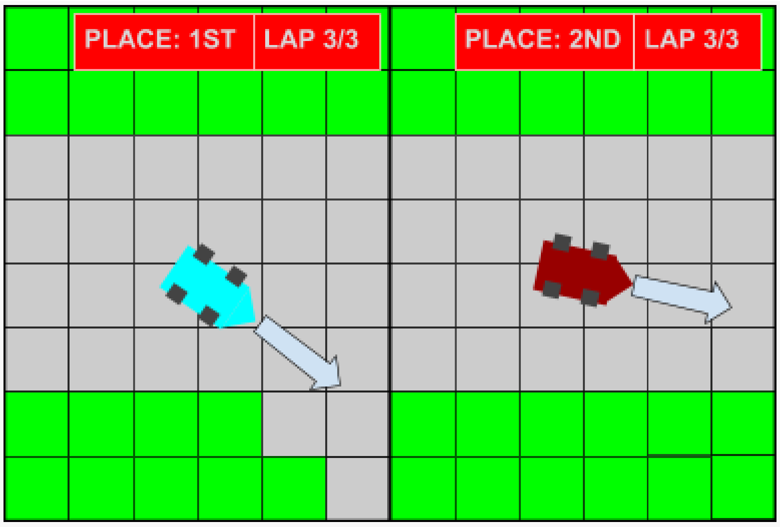


Simple illustration of the menu

AI:
Artificial intelligence (AI) is implemented to increase playability and challenge in of game. By controlling a vehicle object, the AI can maneuver through the track to compete with the player in the race. It is also able to use projectile weapons. As mentioned before, the possible solutions for AI implementation are “port”-objects in the map for AI-cars to follow, or a vector flux field to give a acceleration vector for AI-car in each point of the map.

Projectile:
Vehicles are able to fire at other vehicles. The players should see the projectile coming in order to dodge bullets. The projectile can, for example, slow vehicle down, push, or turn its orientation.

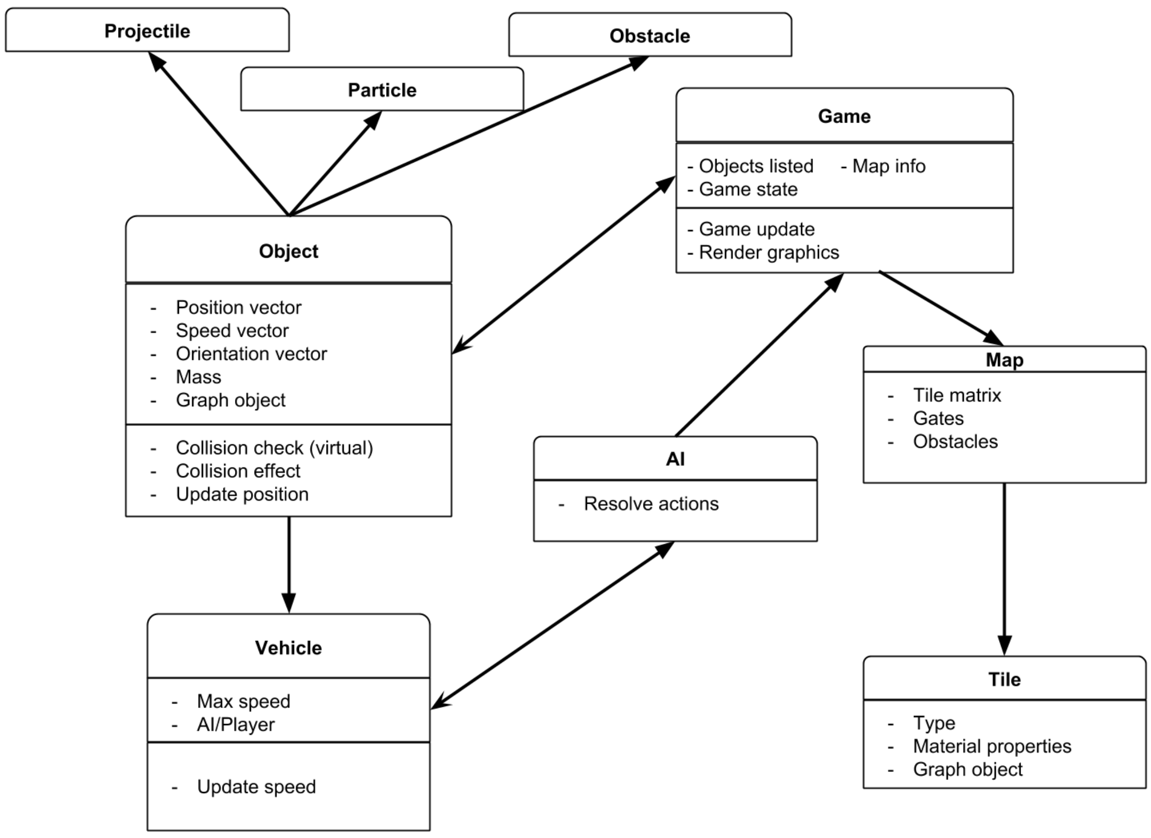
Multiplayer:
The split-screen multiplayer enables game to be played with many players simultaneously. The window is divided to equal sized parts for every player to use.



Mock-up of the splitscreen multiplayer

Sampsa Hyvämäki
Arto Lehisto (PM)
Tuomas Poutanen
Mikko Murhu

UML-class diagram: Micro Mahines



The class diagram shows what classes will be used and how they are connected. To have a class know something about some other class, the required objects are passed to the object as arguments when instantiated. Because all the objects have many common features, we have an abstract Object-class, which from the objects are inherited. The Game-class stores all the information of the game's state and is used for communication between objects. It also has the functions that update the game's state by calling all of the objects' update-functions and drawing the objects by calling the objects' draw-functions. This is why the Game needs to know about Objects and the Objects need to know about Game. The AI-class defines its behaviour by the state of the world, and the state of the vehicle it's controlling, so it needs to have knowledge of both of them. A Tile-object stores information about a fixed size square in the world, how the square is drawn and how an Object should behave when on the tile. A Map-object stores information about the map, basically a matrix of tiles.

Sampsa Hyvämäki
Arto Lehisto (PM)
Tuomas Poutanen
Mikko Murhu

Preliminary schedule

Project daily schedule

Our project work schedule is loosely based on an agile development framework, scrum. We'll have regular check-ups on progress three times a week, where every team member reports 1) what they have done since last check-up, 2) What they'll do next 3) are there any problems or hindrances. In addition to this, we'll have a weekly development cycle, where we fulfill the goals listed for that cycle and produce a working demo of our project.

Preliminary Sub-objectives

20.11. Very simple base for game (makefile!)

- Game draws something on screen
- Input handling somehow
- Player can move things on screen
- Some of the underlying classes implemented

25.11. Basic gameplay with simple driving physics

- Simple graphics that shows everything, simple map
- Input handling
- Moderately fun gameplay

9.12. Other features:

- Multiple players
- Multiple tracks loaded from files
- Game objects which affect to the gameplay (Oil spills, jams, boosts etc.)
- Fun gameplay!