ELEC-A7151
Aarni Kolehmainen, Henrik Granö, Joona Viheriä, Waleed Iftikhar

Air combat game project plan

Scope of the project

The project goal is to create a 2D air combat dogfighting game with simple graphics and physics, where the player pilots a plane against enemy ground forces and planes in different locations.
The player can control the plane by turning the engine on or off, pitching up or down, by firing the machine guns and by dropping bombs, using the keyboard.

The player can play against ground forces who try to shoot the player down, or against another plane controlled by a player on the same keyboard.

Enemies:
Infantry
Flak
Enemy plane

Maps are imported from suitable 24-bit BMP image files, which can easily be created by using a simple image editor, e.g. MS Paint.
The graphics and physics will be implemented using the external libraries SFML and Box2D, respectively.

Rough program structure, class hierarchy

App:

- Main class, creates and keeps track of the window (SFML) and main loops along with event handling
- Main menu and game view: two main member functions, mainloop() which runs the event handler loop for the main menu, and gameloop() which runs the event handler for the game view. Gameloop() calls the class's render() member function to draw each frame as well as the **Game**-class's update() function to update the physics in the game.
- The constructor of this class initializes the window (SFML)

Game:

- Contains information needed for the game session: the map, vector of planes in the game, vector of groundforces and, if implemented, vector of other objects on the map such as hangars
- The member function update() updates the games physics, i.e. locations and collisions for each plane in the game.
- The constructor of this class initializes the Box2D world needed for the game's physics

Master_of_planes

- Abstract class
- Master class for different planes available in the game.
- Common attributes for all planes: coordinates, acceleration, max_speed, hitpoints, associative container of weapons
- move() is called by the update() method from the class **Game** to calculate the flight path

Child classes for different types of planes, inherits Plane parent class

- Planes with different qualities, some planes might have only bombs or machine guns or different values for max speed, acceleration ability or hitpoints

Groundforces parent class for ground forces

- Abstract class
- Common attributes for all ground forces: coordinates, hitpoints, weapons
- Main ability is to shoot at enemies using the shoot()-function, different implementations in different units (pure virtual)

Child classes for separate units, inherits Ground forces parent class:

Flak

- Stationary anti-air unit, fires slow but effective rounds

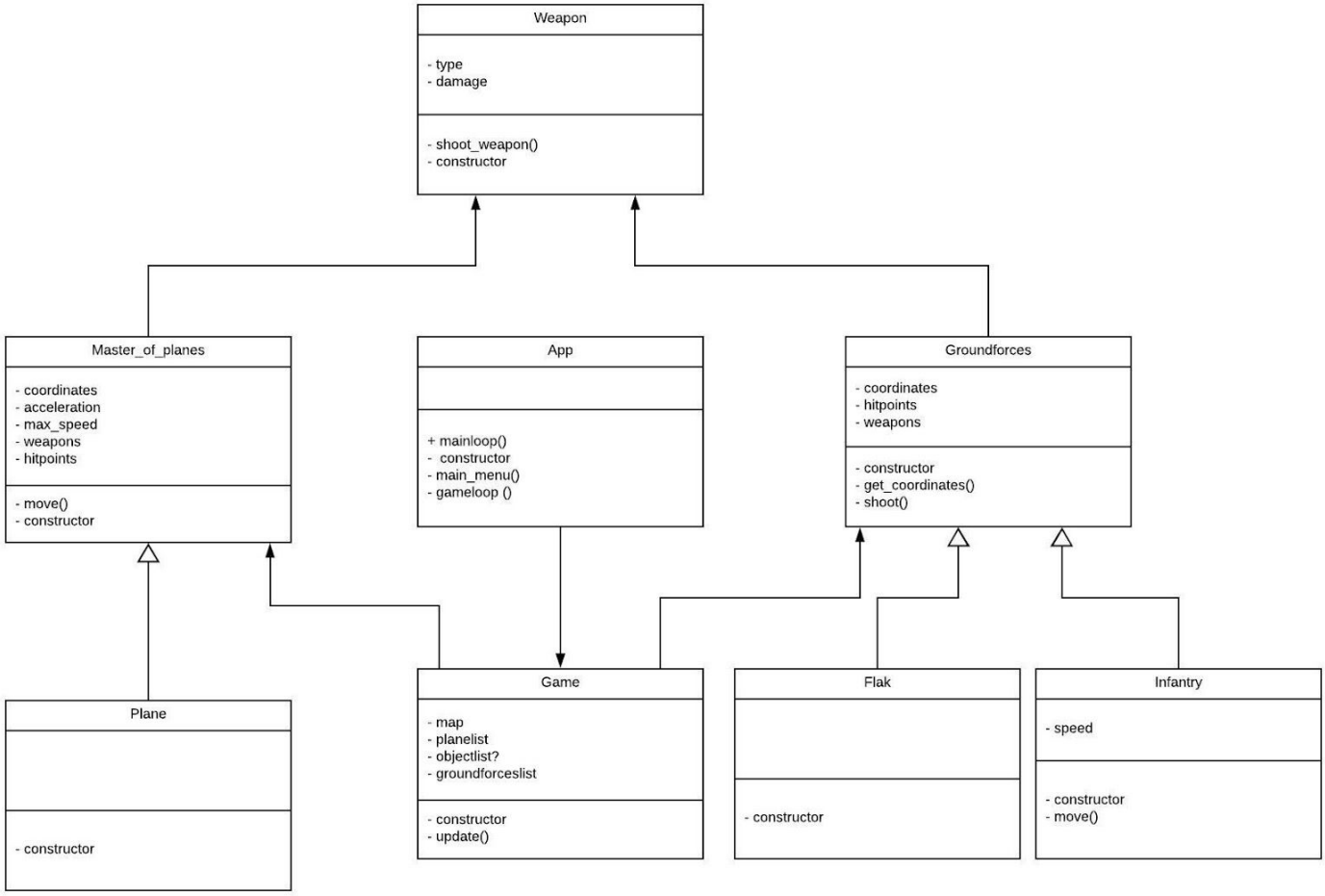
Infantry

- Ground unit that can move and fire small arms rounds against enemies

Weapon

- Used by planes
- Can have different type and damage

UML-diagram



Algorithms

Basic gravity for bombs to drop naturally and for the planes to not be able to stop and still float. Once the plane's velocity drops below a certain threshold, it slowly starts to move downward. Doesn't affect ground units or machine guns since we can assume the speed of the bullets so great that applying gravity doesn't make much difference.

The program applies basic acceleration physics and does not consider vertical air resistance significant. Horizontal air resistance for bombs brings a cool effect as the bomb's horizontal speed decreases while it falls.

Movement example:

```
x_velocity += x_acceleration
y_velocity += g_acceleration
object.move(x_velocity, y_velocity)        # updates coordinates according to speed
```

Collisions are handled by Box2D.

Preliminary schedule

During the project we will be trying to follow the guidelines of agile software development. We have planned 4 sprints each lasting one week. We have agreed upon preliminary goals for each sprint. We are going to have a weekly session after each sprint and see how well we reached our goals for the last sprint and update our goals for the next sprint. We'll also agree on the division of labour in these sessions. Below is a rough outline how we the development work might be spread over the different sprints. This is bound to change quite a lot during the course.

Sprints:

- Working main window, map loading
- Ability to fly and shoot, graphics
- Enemies and collision
- Sound, graphics polishing, local multiplayer, extra

Work distribution

Main logic

- A broad subject that can be divided into smaller components for work distribution

Graphics

Movement

Physics