Major Project Design

Izak Baldacchino a1830164
Hamoun Mohammadi a1716074
Tharane Thamodarar a1787411

https://github.com/HamounM1/Battle_Jets.git

Project Specification - Major Practical

Prop Fighter

Introduction

The project is a turn- based fighting game where you, playing as a fighter jet, have to battle an enemy jet. The plan is to utilise abstract classes which are then inherited into child classes such as the enemy squadron and player squadron. The game is interactive, it's features include the player choosing the type of aircraft they would like to use in battle and the difficulty level of battle (the enemy pilot skill). In this game each round has a few phases and the player initially chooses to commence battle. Each phase the player chooses an action in response to the outcome from the previous play. After each action a result will be displayed of their health and the outcome of the previous action. The player will be presented with a squadron of aircraft and their specifications. The winning conditions are the enemy retreats, severe damage to aircraft or death of pilots.

Design Description

Memory allocation from stack and the heap:

- Arrays: arrays will be used for upgrades, armour, REPAIRS(Maybe).
- Strings: names, boolean, status reports, attacks calls, upgrads.
- Objects: aircraft (enemy and player).

User Input and Output

- User input is used during different phases of the round.
- Choosing upgrades
- Choosing type of aircraft

Code Style

All code will be accessed via github. Either done on our personal laptop CS50IDE or university computers. The naming convention (constant variable will be written in all uppercase with an underscore) in this project will be in Apps Hungarian.
The abstract class is used to inherit information to the child classes where the aircraft specifications, damage control and squadron will be kept. Code comments will be used to briefly describe what each section of code does to reduce confusion and ambiguity among the group. A few additions might occur if time allows such as a store, credit score, visual and sound effects.

Class, function and variable names:
c_air.h - the aircraft class has functions which helps display the statistics on the aircraft for the player to choose.

c_board.h - is the array/game board/ airspace where the battle will take place. This array is dynamically allocated and is used to display, maintain and allocate a position for the player and the enemy.

c_pilot.h/cpp - is the pilot characteristics class. This holds information such as player position on the board, health and has functions which will then be used as input from the players to attack, descend or ascend if nothing is mentioned it will show no action.

game.h/cpp - it is another class file for the main code that is done in the game.cpp file. These include macro code to reduce too much use of memory and space. Has functions such as player action, descend and ascend which complements the pilot class files.

c_enemy.h/cpp - this is the virtual AI opponent the player will face. The enemy's response is randomised according to the player's action. It has similar features to the pilot class like health, position on the board and outputs.

game.cpp - it is where the main code happens for the game to run. It features all the functions and this will run in a sequence where the main file will have a while loop calling game.cpp and when the player wishes to terminate the game the while loop will stop.

<u>Schedule Plan</u>
Goal by the end of week 8 is to come up with an idea, general outline of features and delegation of jobs in the group.

Week 8
- Topic idea
- Plan
- Features in the game such
- Code sharing via GitHub - SVN commit
- Google docs - for the plan document
- Design document - write up classes and variables in the document. Elaborate more on the use of the classes.

  Delegated Components
  - Izak: Class for game.h, aircraft.h and pilot.h
  - Tharane: main.cpp the main functioning of the game. The display messages, the type of aircraft
  - Hamoun: start on enemy class

Week 9

- Basic skeleton of code: the board array working and the positions of the pilots via unit tests.
- Makefile (to run all the code will be using ./main to run the game)
- Testing strategies (unit testing: each class has a test file, running a few variables to test characteristics and check for errors).
- Add more to the design document if there are changes (class diagrams, add more variables and class names to the document).
- Sort out SVN upload to the university submissions file.
  Delegated Components
    - Izak: Class for game.h, aircraft.h and pilot.h
    - Tharane: main.cpp the main functioning of the game. The display messages, the type of aircraft. Design document updates.
    - Hamoun: Enemy class

Week 10

- Have a functioning code displaying board arry and player positions and update Makefile for added files.
- Add in enemy, pilot, aircraft, game class to see how it all runs.
- Upload to SVN
  Delegated Components
    - Izak: Class for game.h, aircraft.h and pilot.h. Design the game class which includes the functionality, the board (size and positions of the board)
    - Tharane: main.cpp the main functioning of the game and game.cpp files such as the display messages, the type of aircraft and the general while loop used in the main.cpp file. (make a while loop, run for different board sizes)
    - Hamoun: start on enemy class (behaviour such as randomised code to counter the players actions)

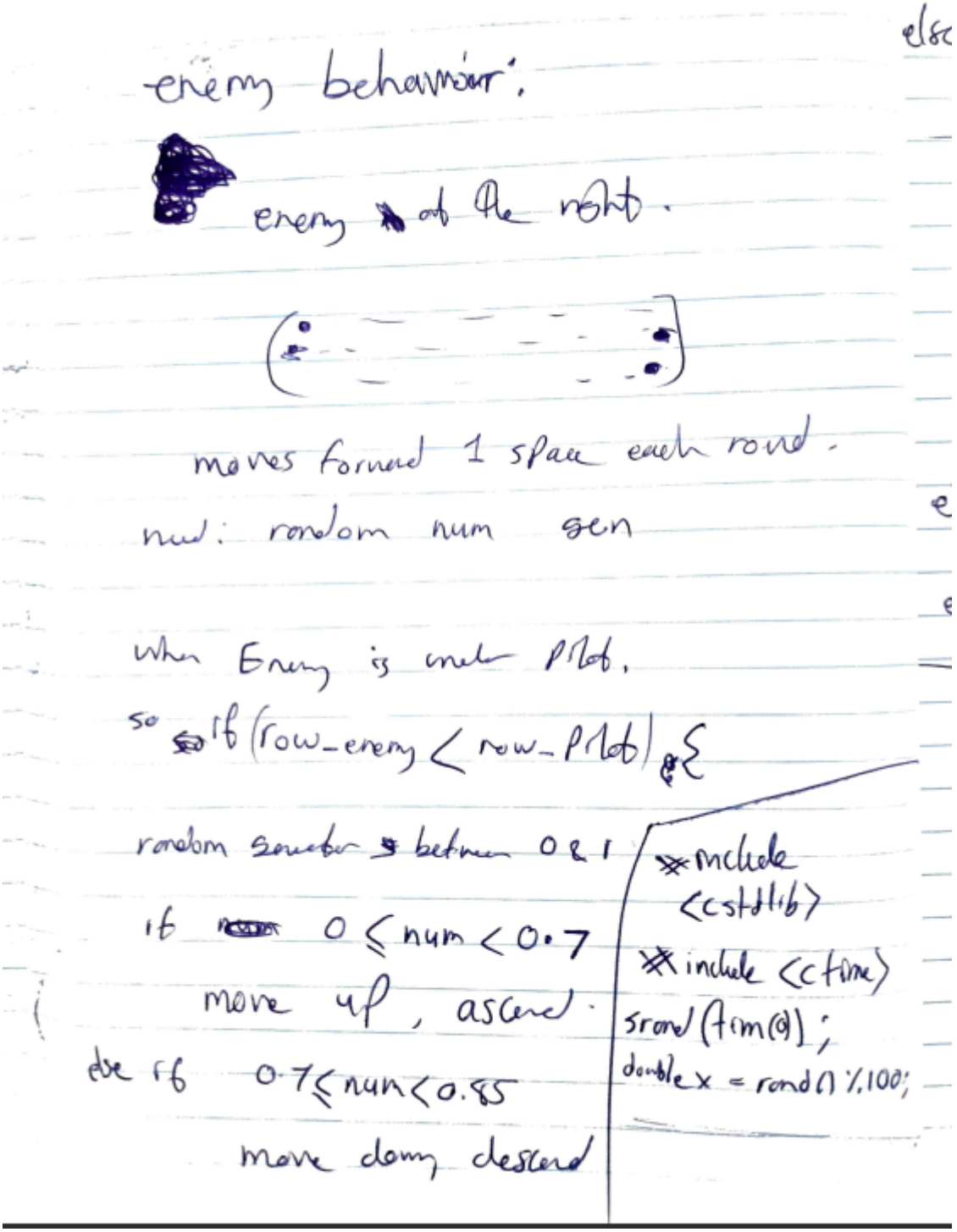
Week 11:

- Have a final copy ready, uploaded to svn
- Present the project with each component explained by Tharane, Izak and Hamoun. Pilot, aircraft, enemy, game classes and show the main.cpp file.
- Final makefile, ensure the makefile has no errors and any additional files have been added to the makefile.
- Have everything uploaded before the practical
-

<u>Testing</u>

Unit testing: each class has an individual test file which tests the characteristics and looks for errors. An example will be shown by the group.

Planning done on paper: Hamoun Enemy class behaviour



else if 0.85 ≤ num < 1
attack!

Enemy above Plot

ex. if (row_enemy > row_Plot) {

if 0 ≤ num < 0.7
descend

else if 0.7 ≤ num < 0.85 ascend

else if 0.85 ≤ num < 1 attack!

Enemy & Plot on same level:

ie if (row_enemy == row_Plot) {

if 0 ≤ num < 0.7 attack!

else if 0.7 ≤ num < 0.85 ascend

else if 0.85 ≤ num < 1 descend

Project:

col P starts at 0,
col E starts at max_col

(P E)

when they pass [E P]

if (colP - colE == 2){
reset board positions }

~~board update~~: board update:

~~if (colP - colE == 2)~~

~~else~~ RowE = PilotEnemy.getRow();
RowP = PilotPlayer.getRow();
ColE = PilotEnemy.getCol();
ColP = PilotPlayer.getCol();

if (~~col~~ P - ~~col~~ E == 2){

PilotEnemy.setPosition(RowE, max col)
PilotPlayer.setPosition(RowP, 0)

Class Diagram