Patrick Sumarli a1811952 - Muhammad Sharjeel Nawaz a1805404 – Ti Wei Lu a1818152 - OOP Semester 1, 2021 - The University of Adelaide

Project Specification - Major Practical

Introduction

Our project implements a virtual arcade for people to play. Players can select which game they want to play, make choices for those games, gain points for the games and exchange points for prizes (assuming they're real life prizes)

Design Description

Assessment Concepts

Memory allocation from stack and the heap

• Variable names: Game.name(string), Game.maxPoint(int), games(Game object pointer dynamic array), Basketball.chances(array of int)

• Objects: Game. Their methods include Game.play(), Game.result(), etc.

User Input and Output

• I/O of different data types: Command-line. Players can enter numbers to select an option from the main menu, enter strings to make choices during the games, etc.

Object-oriented programming and design

• Inheritance: The Game superclass will be inherited by the children classes such as Basketball, etc.

• Polymorphism: The games will implement their own version of Game.play(), Game.result(), etc.

• Abstract Classes: The Game superclass will have a pure virtual function such as the play function to make sure the children classes implement their own versions of them.

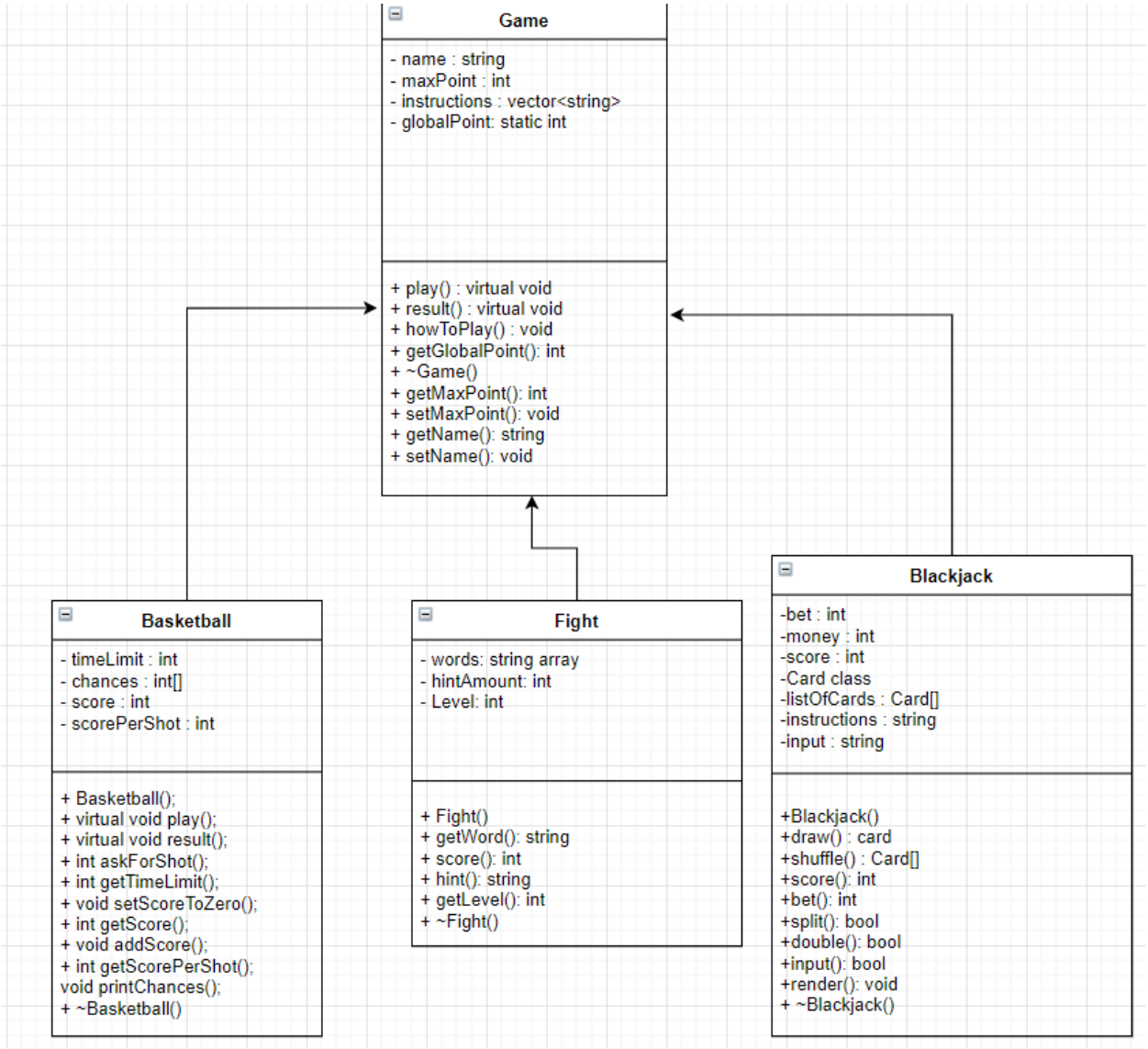
Testing Plan

• Test inputs and expected outputs: We will have a set of inputs and expected outputs to test whether or not the program works as expected. These inputs will include expected inputs, unexpected inputs such as different data types, edge cases, as well as extremely large or small inputs.

• Automated Testing: We will use a Makefile to run automated tests as well as inputs and compare the outputs and expected outputs if necessary using a filenameTest.cpp file for each component in the program.

• Regression Testing: With each feature we add, we will have new tests written for it and we only proceed to the next feature if the tests for the current feature pass. When creating the next features, we will also be running the tests for the previous features to make sure they still work as intended and the new features do not break their functionality

Class Diagram



Class Descriptions

Note: All of the listed states and behaviours are what **must** be in the classes. Extra features could be added but these are the main and most important features.

Game

Abstract base class. The game class will provide a foundation for all the child classes to build on.

Variables:

- name: stores the name of the game
- maxPoint: stores the maximum point you can get from the game
- instructions: stores the set of instructions the player needs to understand the game (used mostly if game has option to show instructions whenever user wants them)
- globalPoint: static int type that stores points from games and adds them together

Methods:

- play(): pure virtual that starts the game and gives starting values to variables
- result(): pure virtual that displays the result of the game
- howToPlay(): display the instructions of the game
- getMaxPoint(): returns the maximum point you can get from that game
- setMaxPoint(int): sets a new maximum point for the game
- getName(): returns the name of the game
- setName(): sets a new name for the game

Basketball

A game based on basketball that inherits from the Game base class.

Variables:

- timeLimit: the amount of time available for the game
- chances: an array that stores the chances for each type of throws
- score: a variable that stores the amount of scored shots the player made
- scorePerShot: a variable that stores the amount of points you get from a shot

Methods:

- Basketball(): Constructor to initialise all the values needed for the game to run
- askForShot(): ask the user for a choice regarding the type of shot they want to make
- getTimeLimit(): returns the time limit of the game
- setScoreToZero(): sets the player score to 0
- getScore(): returns the player score
- addScore(): runs whenever the player makes a successful shot

- getScorePerShot(): returns the score the player can get for each successful shot
- printChances(): prints out the chances for each of the shot types, mainly used for testing

Fight

A game like hangman but the user has to guess multiple words to get through stages of the game. The word gets progressively harder

Variables:

- words: A string array of the words to be guessed by the player
- hintAmount: Amount of hints left for user to use
- Level: The level the player is on

Methods:

- getWord(): method to get a word according to level they're on
- getLevel(): getter for the current level and increments level by 1 each time it is called
- score(): the final score the player has that would be given to the globalPoint variable

Blackjack

The classic blackjack game. Score is calculated when there are less than 10 cards in the deck or if the player runs out of money.

Variables:

- bet: amount that the player bets for the round
- money: amount of money that the player currently has
- score: stores the final score for the player
- rounds: number of rounds the player has played
- input: stores the input from the player
- Card class: class for storing card objects
- Player class: class which contains a vector of Cards and some methods
- deck: vector which holds card objects

Methods:

- draw(): draws a card from the deck.
- score(): Calculates the final score
- setBet(): sets the bet by the player for the round
- splitPlay(): a separate method for when the player splits in blackjack
- doublePlay(): a separate method for when the player doubles in blackjack
- getInput(): validates the input given by the player
- render(): method which displays the cards drawn by the player and the dealer
- dealerTurn(): method which simulates the dealer's turn

User interface

The program user will be a player of the arcade. They use the command-line and will be able to choose from different options available in the program using words and numbers.

Code Style

All code in the program will be properly indented using 4 space tabs.

Variable and function names will be written in camelCase

Classes will begin with a capital letter. They will be separated into two files, one for declaring them(.h) and one for implementing them(.cpp).

Comments will be used to describe blocks of code to explain their purpose and the part they play in the overall program

Testing Plan

Unit Testing

For example, we will have a test file for the Basketball object. This test file will create an object instance and we will print out the variables and compare them with what is expected. Afterwards, we will run the object's methods and compare the results with the expected results.

Automated Testing

Our Makefile will compile and run the test files. We will use the command line makefile commands such as "make basketball" to compile and run the test files for the Basketball object. If the test file needs inputs, we will have a .txt file to automate the inputs and save the outputs to another .txt file to compare it with another .txt file for testing with expected outputs.

Regression Testing

When we code the members for a class like the Basketball class such as the Basketball.maxPoint variable, we run tests such as printing it out to see if they work as expected and contain the data we expect it to. Afterwards, when we create a new class member such as Basketball.result(), we run tests for that new functionality and also tests for the previous members such as Basketball.maxPoint to see if they broke or not. We test the new and existing features to make sure that nothing breaks.

Schedule Plan (Revised)

Stretch Goals

Our goal is to complete the design document and plan by week 9. If we have finished it before the deadline, we will start writing on the code to create the foundation for the program

By week 10, we aim to develop the core features of the program such as the game selection, the Basketball and Blackjack games as well and create test files for each of them. If we were to finish early, we would continue to code the other features of the program.

By week 11, we will finish the program and verify that there are no bugs with exhaustive unit and regression testing.

Week 8 - Preparing for assessment in Week 9

Check-list:

Develop a project plan -Patrick

Create class diagram -Sharjeel

Upload to SVN -Sharjeel

Organise code sharing with group members. -Ti Wei Lu

Establish a testing strategy -Patrick

Write makefile -Sharjeel

Week 9 - Preparing for assessment in Week 10

Check-list:

Code the Basketball game -Patrick

Code the Fight game -Sharjeel

Code the main menu -Patrick

Code the view game history feature -Patrick

Create test files to conduct unit tests on the games -Everyone

Upload to SVN -Sharjeel

Week 10 - Preparing for assessment in Week 11

Check-list:

Code the Blackjack game -Ti Wei Lu

Code the prize feature -Patrick & Sharjeel

Run all test files to make sure nothing is broken -Ti Wei Lu

Upload to SVN -Sharjeel

Compare with design document - Everyone