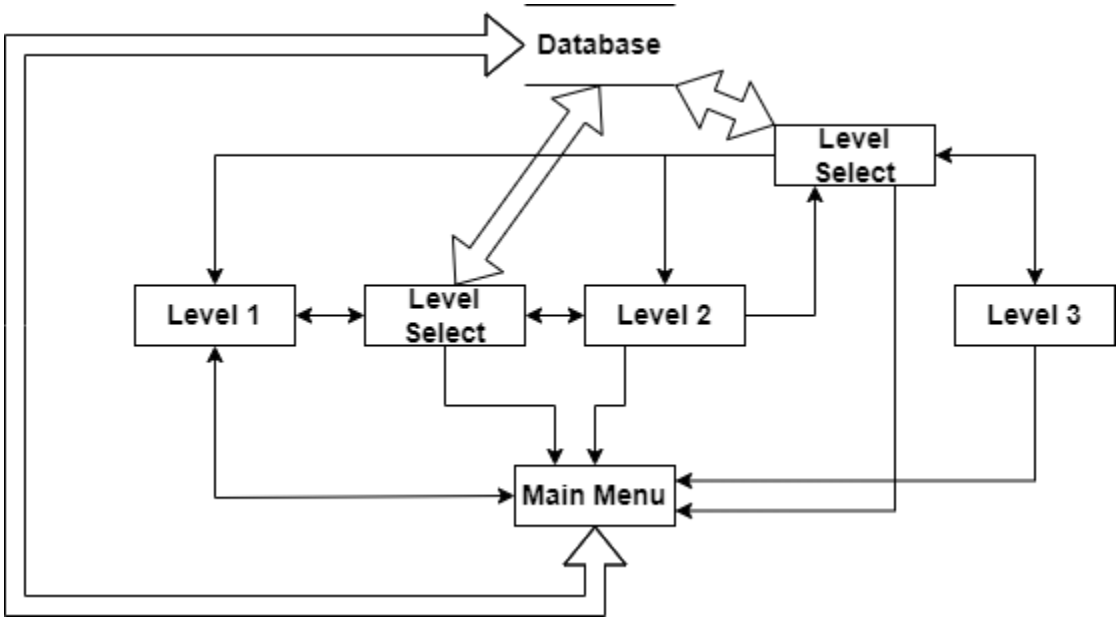
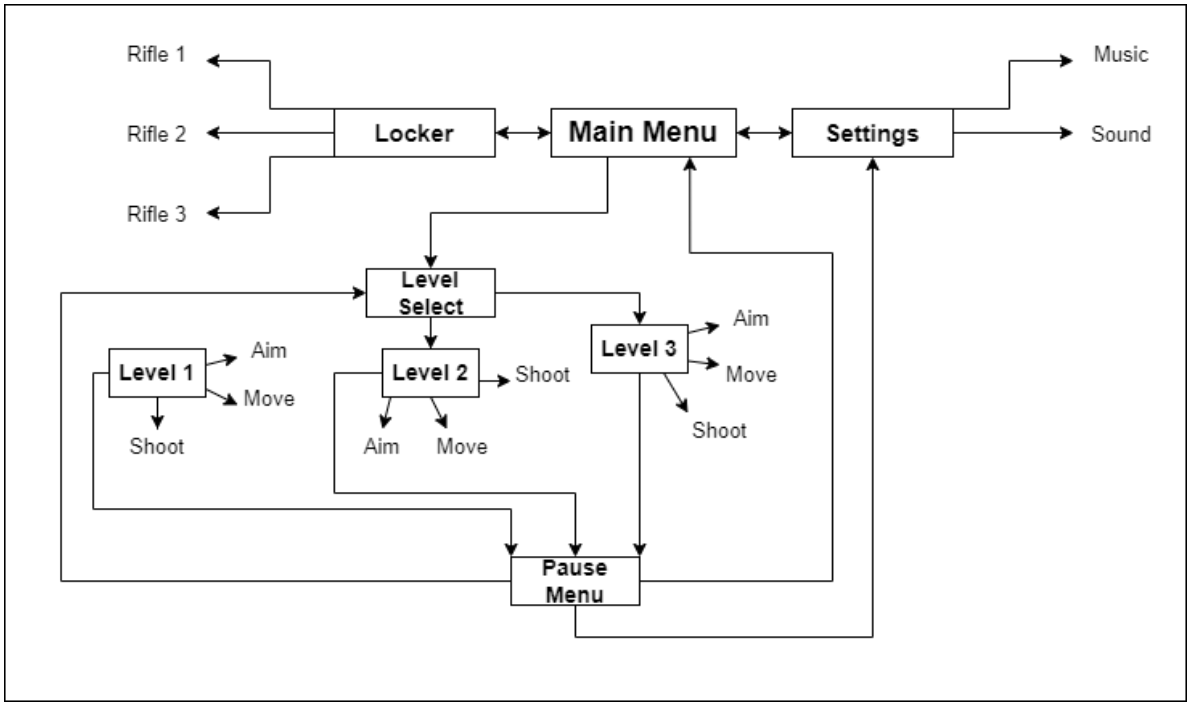
High Level Design

High Level Architecture - Alec York



The above diagram is a combination of the architectural styles: Pipes and Filter, and a Repository. I went with the Pipes and Filter style because it would be apparent that input of main components of the system would essentially be converted into output. For example, completing level 1 gives the user access to level 2 by letting them select the levels between 1 and 2. I also opted to combine this style with a Repository, because repositories involve a number of components (e.g. the level selects) that have the option to update, retrieve, and store data into a repository. This is relevant to our system because we will need to have the option for users to save the game before/after levels, and repositories allow the system to do so. To provide some more context into what the diagram achieves: from the main menu, you can check for any save files to continue the game, or you can start at level 1. After completing a level, the user will have the option to either save their progress, or continue to the next level. At any point, the user will be able to return to the main menu.



Design Issues - Alec York

- The system has to have a reliable menu and options settings. Gameplay-wise, it should also behave in a mostly realistic manner (i.e. everything should work in the way the player would expect).
- The system is a mobile game, so it should have portability.
- Doesn't have to be specific evaluations, can be more abstract. Have the reasoning connect back to the high-level architecture. Example: System is a web-app, so should have portability

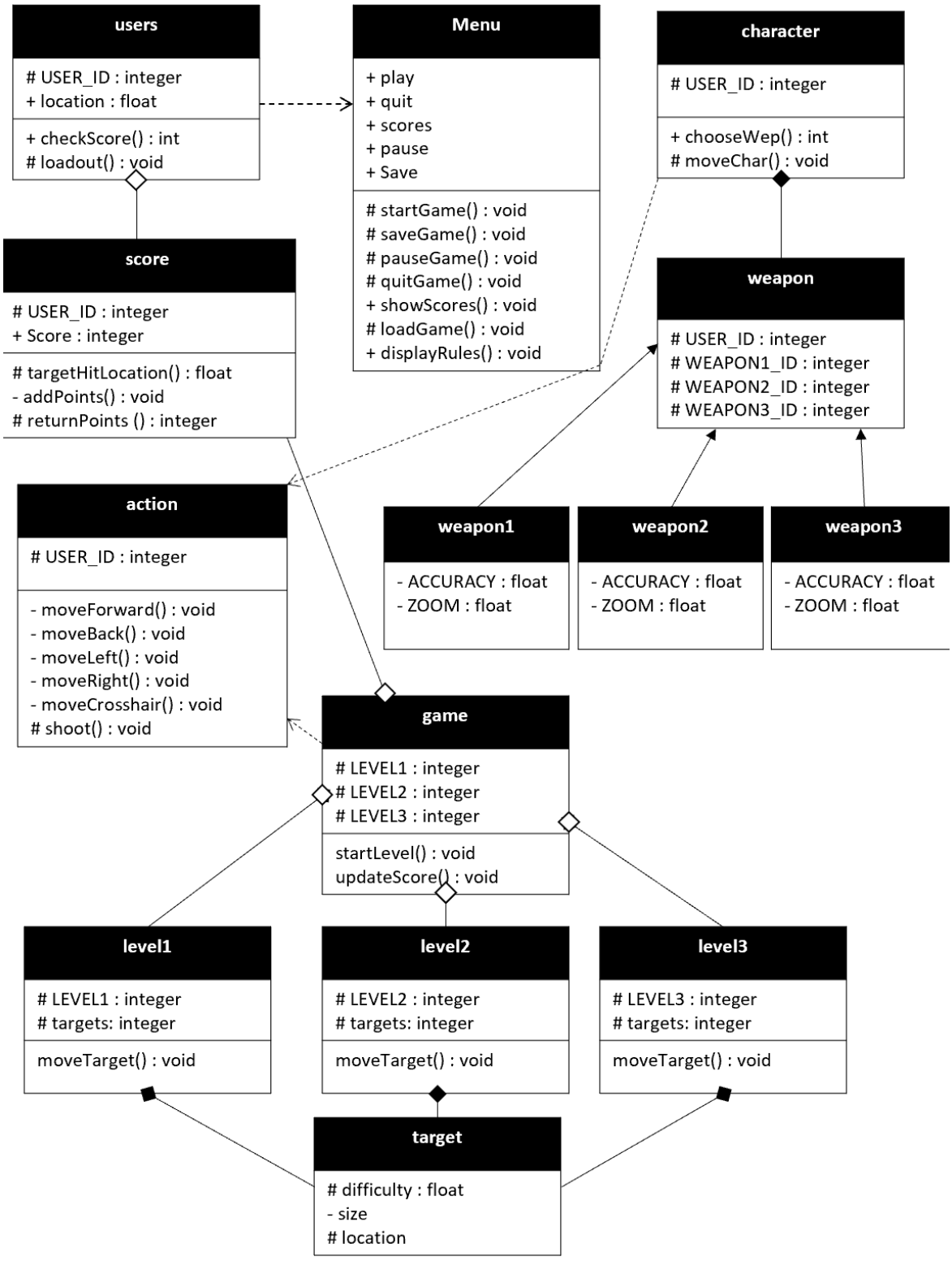
The issues most relevant to this project include: portability, reliability, reusability, testability, performance, and safety. Portability: this architecture supports the portability of the system, as it does not require an online connection, as there is no multiplayer, and therefore does not require a strong internet connection to play. Reliability: the system has to reliably be able to save and load saved data, and a repository works well with this since it allows multiple interfacing components to work with the same data. Reusability: because there are multiple components that are similar within the system, e.g. the level select interfaces (depending on the progress of the user, the number of options will change) and the levels themselves, as they will follow similar formulas and utilize similar code, reusability is supported in this architecture. Testability: there are many things to test for, whether it be testing if the transition buttons to a different interface works properly, or if the game runs in a realistic manner, this architecture makes testing of the system and its components pretty straightforward, as the paths are not too

complicated and can be easily followed. Performance: the performance of the system is also contingent upon many factors, including whether the save option works as it's intended, or if the game runs fast enough, etc. Logically, the performance should work well if the system is built per this architecture, as the save option should work as intended if it's designed following this architecture and there isn't anything in this architecture that will significantly slow the game down, either.

Although high-level, this architecture does support against running into what are more likely (logically speaking) to be the major design issues down the road. Safety, being another common design issue, can also be compromised if, for example, the system loses the users save files. This can create some technical difficulties, as with this high-level design it's not as clear whether or not it will be easily prevented; however, given this design's support for the testability of the system, it will be easier to be debugged/solved. One tradeoff made in selecting this architecture is that if we decide to make a multiplayer option for the game, there are other styles that would be better suited for this; e.g. client-server where users are the clients hosted by the server, or peer-to-peer where users/clients can host other users/clients in the game. In the case that this happens, a prototype of an architecture including one of these styles (depending on how we want to make/structure the multiplayer aspect of it) would be designed as a means to evaluate these alternative design strategies. In this case, security would also become a potential major design issue, as when introducing a multiplayer aspect, it is important to maintain strong security to prevent cheating, stealing of data, etc. Finally, maintainability may also be relevant depending on how much needs to be adjusted in order to support, for example, some new IOS update. However, it is unclear how exactly this architecture can support this, although I would say given the straightforward aspect of the design, it does help the maintainability of the system.

Detailed Design

Class diagram - Ishmael Garcia



Trace of Requirements to Design - Ishmael Garcia

REQ ID	Requirement Desc.	Architecture Reference	Design Reference
FR-01	System shall be a multi platform fully functional mobile first-person shooter game	Whole architecture system	Whole game design
IR-02	Interface shall be fast, responsive and fluid.	Database Level 1-3 Main Menu	startGame() pauseGame() quitGame() Action Class
PER-03	Physical Environment shall be on mobile platform/modern devices	Database	Whole game design
UHFR-04	User and human factors shall have ease of use functions, instruction and stability	Level Select Main Menu Settings	startGame() pauseGame() quitGame() displayRules() Action Class
DOR-05	Documentation requirement shall provide text explanation of game	Main Menu	displayRules()
DAR-06	Data Requirement shall calculate points and retain save states	Main Menu Settings	saveGame() checkScore() showScores() loadGame()
RR-07	Resource Requirement shall describe actors to maintain system and describe resources.	*requirement fulfilled outside of architecture scope (display info in game store or other platform)	*requirement fulfilled outside of architecture scope (display info in game store or other platform)
SR-08	Security requirement shall ensure user privacy, secure data and hide system data	Database	Protected/private variables and methods(-/#)
QAR-09	Quality assurance requirement ensure software stability and ease of access when installing	Database	Coding methodology of whole design

Test Plan

Overall Objective for Software Test Activity - Chris Cao

- We will have to test that the menu and settings work as intended (i.e. saving progress or quitting), that level progression is functional (i.e. there are multiple choosable levels and the level is considered complete when all targets are eliminated), and that all gameplay mechanics do what they are supposed to (i.e. bullets go where the player is aiming, targets take damage when hit, the player can choose which rifles to use).

Description of Test Environment - Chris Cao

- Web Application: Browser Environment (Google Chrome, Firefox, Safari, Internet Explorer)
- Mobile Application: Emulator Environment
- Manual Testing: Own Environment of the Application
- We will be testing with both Unity's built-in environment, including Unity Remote, as well as a more conventional emulator environment. Unity Remote will allow us to test the software in real-time by allowing us to access mobile controls, which will be convenient because we will be making the game on our computers. However, it will differ from playing the game itself on mobile, as the game will still technically be running on a computer; Unity Remote simply streams it on the phone. To make sure it both works and runs well on mobile devices, we will use a mobile emulator, which will essentially be the same environment as the one in which our software will operate on. The testers will just be us, the developers.

Stopping Criteria - Christopher Robertson

- Time Related: If we find no errors, we'll stop testing a stage after 2 hours. If an error is found, we will continue testing for 30 minutes if we find no additional errors after the fact (assuming the error doesn't cause the stage to crash).
- Success Related: The test will end if all the major requirements are fulfilled with no other critical errors found.
- Good Enough to Deliver: The game will be good enough to deliver if the major requirements, which include: at least 3 functioning levels, one or more non-stationary targets on each stage, working headshot point modifiers, 3 rifle options prior to starting the stage, ability to save/load progress before and after completing a stage, sounds, visual fx; etc, are all working without too many errors occurring outside of these.
- If we find no errors during a test, we will re-run the stage/program at least twice to see if we still experience no errors. If there are no errors present after re-running the stage, we'll conclude that the stage is Good Enough to Deliver. We'll then do at least one full playthrough of the program and the first three stages in order to conclude if it is all Good Enough to Deliver.

Description of Individual Test Cases - Christopher Robertson

Test Case ID	Test Objective	Test Description	Test Conditions	Expected Results
TC-1	User starts the game without signing in.	The user starts the game, doesn't sign in to the game, and is able to start a new game.	See Test Environment	The settings should be set to default values and there should be no save data.
TC-2	User starts the game and creates a profile.	The user starts the game, and selects an option to create a new profile set with a username and password.	See Test Environment	Initially, the settings are set to default values, and there should now be an option to save settings and progress.
TC-3	User starts the game and signs into their profile.	The user starts the game, and signs in to their user profile with their associated username and password.	See Test Environment	The settings should be set to whatever the last changed settings are. There should be an option to load any previous save data.
TC-4	User loads their previously saved data.	After starting the game and signing in to their profile with their username and password, the user loads their last saved game state.	See Test Environment	After loading their save state, the user can continue from their previously made progress in the game.
TC-5	User changes the in-game settings.	The user changes the in-game settings like the fx volume, gun shot volume, and camera sensitivity.	See Test Environment	While the game is running, the changed settings should persist until the game is closed. If the user is signed in, the settings should be saved to the user's profile.
TC-6	User chooses their setup before starting a level.	Before starting a level, the user chooses their weapon to be used.	See Test Environment	Once the user chooses their setup, they should be able to start the level with their selected setup.
TC-7	User starts and completes a selected level with their setup.	With their selected setup and settings, the user is able to move around the stage, aim, and fire their weapon.	See Test Environment	Once a stage has been selected, the user will start that level with their selected weapon, and complete the stage after either taking out the targets in the stage or running out of ammo.
TC-8	User's level score is saved.	Save the user's score once the level has been completed.	See Test Environment	After the stage is completed (meaning the targets have been taken out, or the user is out of ammo), the user's score should be stored in a leaderboard. If the user is signed in, the score should be saved to their profile.
TC-9	Logged-in user saves their progress.	Once the selected stage is completed, the user (if they are signed in) is able to create a new save/overwrite a previously saved data file.	See Test Environment	The user should save their data and be able to exit the game and load the last saved data file on their profile.

TC-10	User exits the game.	The user should be able to exit the game before/after starting a level, as well as during a level.	See Test Environment	When exiting the game the user should be able to save their data (if they are signed in) before finally closing the game, if they are not currently playing a level. If they are playing a level, there will not be an option to save their data.
TC-11	No access to data when not signed in.	The user should not have save/load data or a score saved when not signed in to a profile.	See Test Environment	When the user is not logged in, they won't be able to save/load any data. If they have not completed a stage yet, they will not have a score.

Traces of Individual Test Cases to Requirements - Christopher Robertson

Requirement ID	Requirement Description	Test Case Reference	Status
FR-01	The system shall: Run on any mobile platform | Have 3 levels | Have moving targets | Have level system | Have varying point system | Have various weapons | Have responsive sound | Have background music | Have load/save state | Have high score tables	TC1 - TC10	In Progress
IR-02	The interface shall: Be responsive | Be fluid | Display touchscreen controls | Have high quality visuals | Have minimal tendency	TC7	Succeeded
PER-03	The Physical Environment Shell: Be on mobile platforms | Run on most modern devices	TC1 - TC10	Succeeded
UHFR-04	The user and human factors shall: Have easy usability of the system functions | Need basic instruction to explain game | Not be able to disrupt/break system	TC1 - TC5	In Progress
DOR-05	The documentation requirement shall: Provide text explanation of the game	TC5	In Progress
DAR-06	The data requirement shall: Calculate points to be outputted | Retain previous game data if save	TC5 - TC9	Succeeded
RR-07	The resource requirement shall: Describe actors to maintain/build system | Describe resources to work on project	TC1 - TC11	In Progress
SR-08	The security requirement shall: Ensure user privacy and protection | Securely access and store user data | Ensure system data is hidden/inaccessible from user	TC2 - TC4, TC11	In Progress
QAR-09	The quality assurance requirement shall: Ensure software will never crash | Ensure software is streamlined | Have easy installation | Follow security protocols	TC1 - TC11	In Progress

Appendices

Weekly Progress Reports

Alec York, Christopher Robertson, Chris Cao, Ishmael Garcia

Project 7: Mobile Sniper Game

COP 4331 SPRING 2022

This document contains the weekly reports for the team (either cumulative or individual reporting) for the weeks working towards the submission of each deliverable. If cumulative report format is chosen, please make sure to include names of the team member(s) associated with each task.

Semester week # (include dates Monday through Sunday): ***3/27(sun) – 4/2(sat)***

<The weekly reports should include either a cumulative team progress report per week or individual team members weekly progress>

Cumulative team progress report

What are the completed accomplishments?

- Completed Deliverable 2 documentation
    - Entire High level architecture segment done by : **Alec York**
    - Detailed Design Class Diagram done by : **Ishmael Garcia**
    - Entire Test Plan segment done by : **Chris Cao & Christopher Robertson**
- Further research and development of game framework/design and implementation

What were the issues encountered (solved vs. unsolved)?

- Some issues were structuring the design and architecture of the software. Deep diving from a simple instruction such as character/user movement to actually designing a structure for the instruction (I.E: multiple subclasses for omnidirectional movement while separating it from weapon movement (zoom/aim) for multiple weapons).
- An issue to still work on is layout/design of levels

What are the plans for next week?

- Structure schedules to implement design structures from first 2 deliverables
- Further research of coding methods for game design