Software Development Report

Overview

Project Title: Blackjack Trainer

Project Description: A fully functional Blackjack game with built-in mechanics for strategy and card counting training.

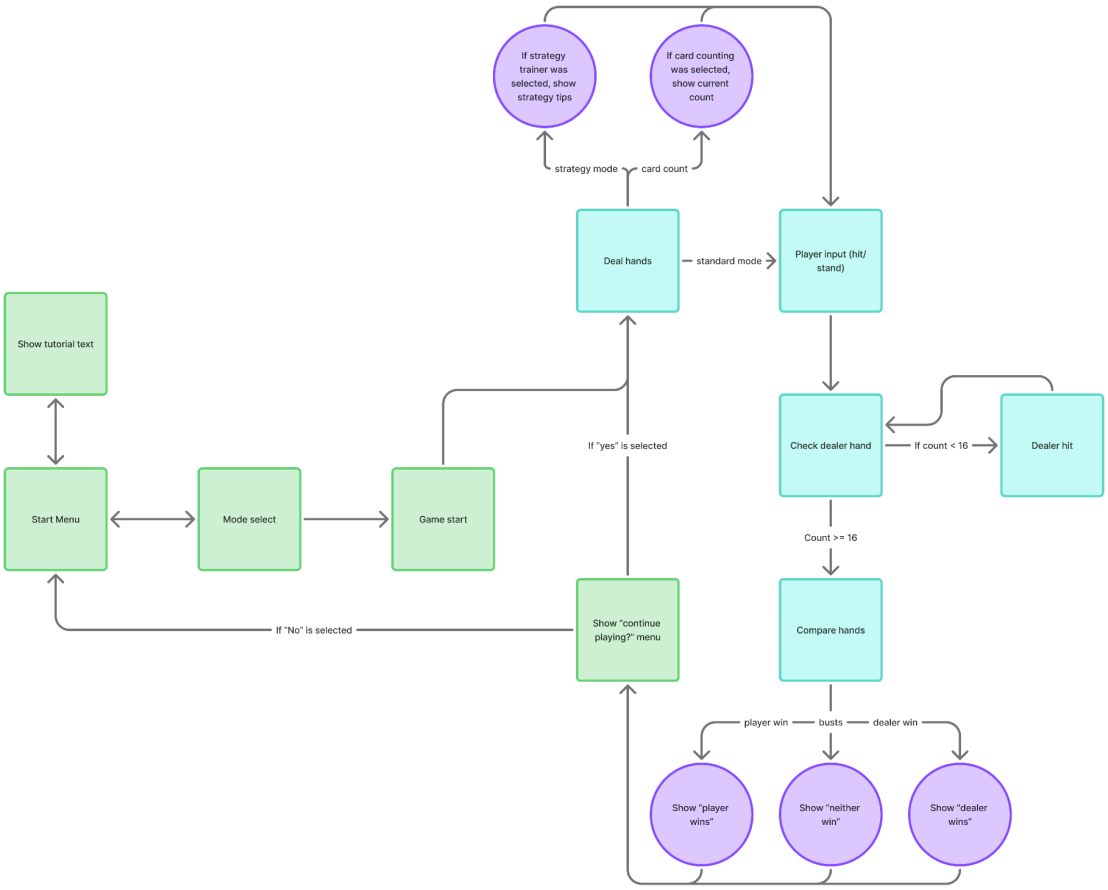
Problem Solved: Software provides an interactive method of learning general Blackjack strategy and introductory card counting concepts, as many online resources that are available are non-interactive (spreadsheets, videos, etc.).

Team Members and responsibilities:

- Logan Morsman - Software engineer, Unity developer, and programmer
- Kai Garcia-Curran - Designer, asset creation

Engineering Planned Approach

This software was designed with an AGILE/Xtreme-Programming framework method in mind. AGILE and XP were followed due to their flexibility in documentation and requirements. They allowed us to modify requirements as needed, and let us focus on documentation at the end of the project. AGILE made it easier to schedule the project, since we could break the major requirements groups into sprints, and work on scheduling the sprints rather than each individual requirement. During the development of the software, we used Unity Editor 6.2 for the game engine, Visual Studio Community 2022 for scripting, and Figma for diagram creation. We also used Github for repository management. The final build is published to itch.io.



Software Functionality Developed

Language: C#17

Functionality: Functions are used throughout the scripts to implement functionality. Detailed descriptions on all functions in all scripts can be found in the developer documentation. A brief overview of full software functionality:

- Main Menu with options to see a tutorial, play game, or quit game
- Play game menu with options to select standard, strategy, or card counting modes

- Standard mode with no strategy tooltips shown to player
- Strategy mode which analyzes player's hand's score, score type (hard/soft), dealer's visible card, and determines whether the player should hit or stand
- Card counting mode which tracks the current card count on the screen
- Hit/stand functionality which can let the player hit to add more cards to their hand, or stand to end their turn
- Casino style dealer automated player
- Cards are shown on screen
- Deck shuffles when it is low on cards and resets the card count
- Shows the winner of each hand after dealer is finished playing
- Offers to return to main menu or continue playing after every hand

Development Process and Completion State

In our traceability matrix/requirements spreadsheet, there is a list of requirements for the final design of the project. Some of these requirements were absolutely necessary for completion of the project, and some of these requirements were stretch goals, in the event that we had an excess of time after completing the base requirements. The criticality of each initially listed requirement is listed here:

Requirement	Team Member	Criticality	Status
Build Github Repo	Logan	Required	Complete
Deck Data Structure	Logan	Required	Complete
Hand Data Structure	Logan	Required	Complete

Player Hand Data Structure	Logan	Required	Complete
Dealer Hand Data Structure	Logan	Required	Complete
Deck Asset	Kai	Required	Complete
Menu Asset	Kai	Required	Complete
Card Asset	Kai	Required	Complete
Game Flow Logic	Logan	Required	Complete
Hit & Strand	Logan	Required	Complete
Main Startup Menu	Logan	Required	Complete
Game UI Asset	Kai	Stretch	Incomplete
Audio Asset	Kai	Stretch	Incomplete
Game UI Logic	Logan	Required	Complete
Strategy Tooltips	Kai	Required	Complete
Mode Selection Menu	Logan	Stretch	Complete
Standard Mode	Logan	Stretch	Complete
Strategy Mode	Logan	Required	Complete
Tutorial	Logan & Kai	Stretch	Complete
Card Counting Mode	Logan	Stretch	Complete
Publish	Logan & Kai	Required	Complete

Stretch goals were audio assets, fancy UI assets, standard mode, card counting mode, and mode selection. Modes were considered stretch goals because the initial intent of the project was just to have a functional strategy mode which acts as the “trainer” in the Blackjack Trainer title. The modes did not take too much time to implement, and so we were able to create the different modes.

First, the standard mode was created. This was because it was part of the initial “Game flow logic” requirement. Completing that requirement meant that 90% of what was needed for standard mode was complete, and I decided to implement it. This is just plain Blackjack with no strategy or card counting tooltips. Once we determined that there we would have enough time to create the multiple modes, we determined that the next best steps would be to get the menus working. This is critical, because without functional menus there would be no way for the user to select which mode they wanted to use. After completing the mode selection menu stretch goal, I created the strategy mode (note, the only required mode with our initial goals in mind). After this, we determined there was enough time remaining to work on implementing the card counting mode as well.

With these stretch goals completed, there were a few left. These were Game UI Assets and Audio assets. The first goal just means overhauling the menus, buttons, and background that the games use to give them a better feel. Currently, the project uses Unity’s built-in tools for asset creation which are extremely limited and they have a somewhat rough look. However, this was not strictly necessary for the functionality of the software. This is similar to the audio asset stretch goal. The game has no audio currently. This is fine, since audio is not necessary for the functionality of Blackjack but it would have been great for aural feedback to the player while they were playing. Instead of expending time on these resources, we instead dedicated our remaining time to polishing what was already there. This included restructuring parts of the UI, fixing bugs which had been found, and changing small pieces of code here and there.

If we were to pursue this project further, the next requirements I would implement into the software would definitely be the audio assets and a UI overhaul. Having the software more visually and aurally appealing would provide a greater chance of increasing the user base. I may

also make further changes to how the dealer works. Currently, the dealer plays very quickly on their turn. Their card is shown, their cards are drawn (if needed), and the post-round winner is announced all in one frame. This is somewhat jarring to the player, though they can keep up with it after some use. Alternatively, there could be small delays in between each step (maybe 1-second sleep interrupts) to make it easier to process how the dealer is playing for the player, and how to keep up with card counts.

We decided very early on in the initial design of the project to have the final software be published as a web build. This meant not having the software be an executable which requires installation on user systems. After development, I believe this was the best possible decision we could have made for a few reasons:

1. Portability - The software runs on Windows, Linux, and MacOS without the need for installation
2. The software can be published and run in websites without us having to maintain a website for it.
3. Building the software can take a while even if it is a smaller project. Building one web build and publishing it is significantly quicker than building for Windows, then building for Linux, then building for MacOS.
4. We do not need to document how to build for every build case, and we do not need to write three different user manuals for Windows, Linux, and MacOS.

Configuration Management Software Metrics

Documenting the run time of the software is somewhat difficult- this varies greatly on the system the software is running on and since the code can potentially execute indefinitely (this is

a game after all, we have no control over when the player quits using the software) the run times are hard to determine. What I can say is that the software was optimized to the best of my ability and is generally very fast. Code is only executed as a direct result of player input, meaning there is no code running continuously while the player is waiting to make a decision. This minimizes resource utilization.

Line counts for all scripts are listed below. You can find more detailed descriptions on the scripts in the developer documentation.

Script	Line Count
GameManager.cs	301
ModeTracker.cs	34
MenuManager.cs	95
Card.cs	21
Deck.cs	53
Hand.cs	142
HandDisplay.cs	73
TOTAL	719

Some scripts were longer than they currently are in their initial implementations, but I have simplified them to be shorter. This makes them easier for programmers to read while maintaining the same functionality. More lines does not necessarily mean better code, or more mechanics.

Licensing

The project is open source with MIT OSS license. More information can be found here:

https://github.com/tippacanoe/Blackjack-Trainer/tree/main?tab=MIT-1-ov-file#readme