Casino Games: A Computer Application Game

Brady Randall, Seth Schoming, Bill Drescher, Justin Hickey

Department of Electrical Engineering and Computer Science

Wichita State University

{bmrandall, slschoming, wjdrescher, jthickey}@wichita.edu

Abstract

This report details the process of design and implementation that TeamWu went through to develop its Casino Game Application. This is an application which provides entertainment through graphical user interfaces via game play in four different casino style games.

1. Inception Phase

1.1 DESCRIPTION

This paper was prepared by Justin Hickey, Bill Drescher, Brady Randall and Seth Schoming undergraduate degree students at Wichita State University in Computer Engineering and Computer Science. The purpose of our project was produce a computer or laptop application game simulating casino game play. Our Casino Game application contains four games. They are Roulette, Black Jack, Slots, and Poker Dice. A player begins game play by entering their name and then choosing which game he or she wants to play from a menu. The games will all operate off of a single bank roll or amount of cash.

The Casino Game application was written mostly in Python 3, but the games developed in later iterations were written in Java. The application began as a list of requirements and then we all worked together to create user stories that would encompass our entire Casino Game application. The user stories were then split up, and each of us four began our work.

1.2 REQUIREMENTS

-Allow user to select which game they want to play; Roulette, Blackjack, Slots, or Poker Dice.
-Display amount of cash available, and allow user to choose how much they want to wager.
-The round or hand of the game is executed and a winning or losing status is determined.
-An appropriate addition or deduction is executed to the player's amount of total cash.
-A user will be able to select if they want to play the same game again or quit and play a different game.

The requirements above were used to create the user stores below.

1.3 USER STORIES

POKER DICE USER STORIES

Choose Opponents

A user will be able to choose to play against one, two, or three computer opponents.

Roll Dice

A user will be able to roll up to five, six-sided, dice and read the values of the top faces.

Second and Third Rolls

A user will be able to choose which dice to keep and which to re-roll. The user gets up to three rolls.

Opponent Logic

Computer opponents will have a basic set of rules to follow when choosing to keep or re-roll the dice.

Scoring

Numerical values will be assigned to the dice combinations of the user and opponents for comparison purposes; the better the hand, the higher the score.

Compare Final Scores

The user's final score will be compared to the final scores of the opponents and results will be displayed.

Menu

There will be a menu at the beginning and end of each game with options such as "Play" and "Exit."

SLOT MACHINE USER STORIES

Slot Machine Logic

A user will be able to start the slot machine program and read an output as to whether or not they won.

Slot Game Menu

A user will be able to choose from a menu of game play options prior to starting the slot machine in action.

Slot Graphical Interface

Upon pulling the lever of the slot machine a user will visually watch the slot machine reels spin and stop on the final figures.

Pay Table

A user will be able to view the pay table for the slot game. The pay table will display what figures in what orders are winning matches.

Multiple Winning Lines

The graphical interface will be larger in order for a user to be able to select more than one winning line

ROULETTE

Roulette Logic and Rules

A user is able to play the game with basic rules and logic, such as one bet at a time and winning or losing at differing odds.

Implement all bets

The user will be able to place all bets possible in Roulette.

Roulette GUI

A user is able to navigate through bets and game play via clicking on buttons in a GUI.

BLACK JACK USER STORIES

Create/Shuffle Deck

The computer logic will create card structures and randomize the deck array before game play.

Deal Cards

Two cards will be dealt to the player and two cards to the dealer, one of which will be face down.

Play Hand

A user will be able to play versus the dealer. Once the cards are dealt, the player can choose hit or stand until he/she busts. Dealer gets similar treatment.

Calculate Scores

The user will be able to see their total score and compared it to the score of the dealer.

Split Hand

A user will be able to select to split their hand if the first two cards are identical.

Accept Bets

A player will be able to enter the amount of bet which could be added or deducted from his pocket.

Pay Out to Player

Upon completion of a round a player will be able to see the calculation of odds of the bet(s) and add to player's bank.

MENU

Graphical Interface

User will enjoy a consistent visual experience across all user stories mentioned above.

Syncing with the Other Games

A user is able to play all games on a common amount of cash, so that winnings from one game carry over to another game.

Table 1: Time Estimates

User Story	Days
Choose Opponents	3

Roll Dice	6
Second/Third Rolls	9
Opponent Logic	14
Scoring	3
Compare Final Scores	2
Menu	3
GUI	14
Total	**54**
Slot Machine Logic	10
Slot Game Menu	8
Graphical Interface	14
Pay Table	6
Multiple Winning Lines	14
Total	**52**
Logic and rules	18
Bets allowed	**10**
Roulette GUI	**15**
Get all games connect	**5**
Total	**48**
Create/Shuffle Deck	3
Deal Cards	4
Play Hand	10
Calculate Score	5
Split Hand	4
Accept Bets	3
Pay Out to Player	3
GUI	14
Total	**46**

OVERALL TOTAL	**106**

ITERATION SCHEDULE : by user story

Iteration 1

- Choose Opponents – Poker Dice
- Roll Dice – Poker Dice
- Second/Third Rolls – Poker Dice
- Slot Machine Logic – Slot Machine
- Slot Game Menu – Slot Machine
- Logic and rules - Roulette
- Create/Shuffle Deck - Blackjack
- Deal Cards - Blackjack
- Play Hand - Blackjack

Iteration 2

- Opponent Logic – Poker Dice
- Scoring – Poker Dice
- Compare Final Scores – Poker Dice
- Graphical Interface – Slot Machine
- Bets allowed - Roulette
- Calculate Scores - Blackjack
- Split - Blackjack
- Accept Bets - Blackjack
- Pay Out - Blackjack

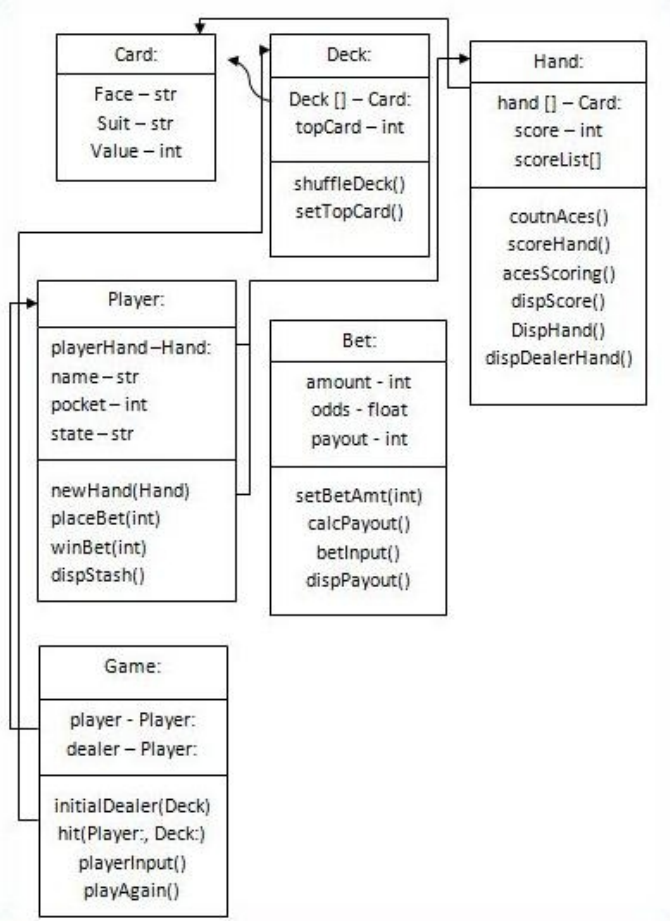
Iteration 3

- Menu – Poker Dice
- GUI – Slot Machine
- Pay Table – Slot Machine
- Multiple Winning Lines
- GUI - Roulette
- Sync - Menu

- GUI – Blackjack
- GUI - Menu

2. Iteration 1 & 2

2.1 UML DIAGRAM



The UML Class Diagram show above was developed for the blackjack module. It is the last of several iterations that were prepared before coding began. Creating this diagram proved very useful in clarifying the interrelationship of objects with each other, and deciding which methods should go with what objects to avoid coupling of objects. Breaking the blackjack module into so many objects also promoted cohesion of the module. As coding progressed into the second iteration (and the rules of the games were read more carefully), it became clear that the decision to make the bet object part of the game object was a mistake. When the games would be coded to allow multiple players, it becomes obvious that a bet object should belong to each player, rather than belonging to the game. Also, two of the games (roulette on all spins and blackjack after splitting a hand) let a player have multiple bets on one round of play. For these cases, it is necessary for the player to have to have a list of bet objects, although much of the time that list would have only one bet object in it.

3. ORGANIZATION AND COMMUNICATION

3.1 Git and Github

We originally chose Git for use as a software repository and version control, but had a great deal of trouble getting it to work properly. After several weeks of frustration with Git, we switched to using Github, which is Git with a central repository and a graphical interface for Windows and MacOS. This worked better as a collaboration tool, but still left the team members complaining about the difficulty of using it. The need for version control on software projects is self-evident, especially if the code is large, long-lived, or undergoing rapid changes (as it would in an agile development environment). But turning untrained users loose on a confusing system is not a recipe for success.

3.2 Team Organization

Our team was not hierarchically organized and the decision making process was collaboration and consensus. While this is probably typical for students working on a one semester or shorter project, the team may have been better served by

adopting a more formal structure with defined and separate roles. Our project at first seemed ideal for a team effort, with the four games making for a natural partitioning of the work, to be followed by a simple (we thought) integration into one program. In hindsight, more effort should have, been expended up front, in program architecture and design. It would have been of great benefit to spend more time thinking about what objects and methods should be common in all of the games. Perhaps simple common classes could have been defined, with each game programmer using inheritance to tailor them for his specific game situation. Also more thought should have gone into the eventual addition of graphic elements into each game.

4.PLANNED WORK ON ITERATIONS

For our project we planned on having a casino with 4 games. Poker dice, roulette, blackjack, and slots all operating under a singe menu. We wanted all the games to have graphical user interfaces that made the games enjoyable for the user. We wanted the user to be able to enter their name and enter how much money they had with them when they went in our casino. We wanted all the games to share a single money pool that would increase or decrease based on the users performance on each game and which would transfer from game to game.

5.ACTUAL WORK ON ITERATIONS

Our group was able to get a portion of that done. We didn't anticipate so many problems with pythons tkinter library. Many of our issues came from tkinter because Brady was really the only one who somewhat knew tkinter going into this. This caused a large amount of time just trying to teach ourselves how tkinter worked and how to manipulate all the functions associated with tkinter.

Another big time consumer was to try to get a wrapper for java to python and how to use GUI's with that wrapped code. We started to research wrappers but then we really didn't see a lot that helped us out so Justin and Seth tried to convert all of Seth's Java code to python which was really time consuming. This made us feel like we were just going around in circles so we switched back to using a wrapper. Unfortunately we were unable to implement the wrapper before the end of this class.

All that being said we feel that we still accomplished a lot as a group. We were able to get all the games functioning in command lines. We were able to get a GUI to work on blackjack, we got a GUI on roulette, we got a somewhat functioning GUI on slots, we got a menu that works well with roulette and blackjack, we had slots almost working with the menu, and we got poker dice to have multiple CPU players that were somewhat competent. We also learned how to build test cases, UML diagrams for the games we made, and learned how to work as a software engineering team.

6. CHALLENGING DESIGN FEATURES

The most challenging features for our programs were the GUI's and getting a wrapper for java. These both took so much time to try to fix that if we had an option next time we would most likely use a different GUI and try to use only one language. Another challenging issue was trying to get the CPU players in poker dice to think like a real person would think. This was challenging because it is hard to try to simulate a humans way of thinking on a computer. We spent a lot of time just trying to think of how the computers should figure out which cards to pick and which cards they should drop. Every time we thought we had it working then something else would go wrong so we just decided to make it simple for iteration 2 then in our next iteration make the computers more complex.

Another problem we had was trying to integrate all the programs into one. This was tough because we sometimes had exactly the same variable names so the program would freak out. Brady had the menu all coded so we all just added our code to his menu but we soon found that our programs were breaking each other. First Brady got his code working in the menu then we took Bills code and added GUI's to it and got it running by itself perfectly. When we put Bill's code into the menu his code all of a sudden didn't work and we still haven't figured out why. We then got Justin's GUI's working and then put his code into the menu even though we didn't completely fix Bill's program. Well when Justin's code got put in the menu it completely broke both Brady's and Bill's code. This made connecting all the code together VERY frustrating. We already knew combining the code would be a nightmare but we had no idea that it would be that challenging.

Our final challenging was getting code working on linux, windows, and mac. This was tough because tkinter had tricks that only worked in linux but when someone on windows tried to run their code it wouldn't work. This was very frustration because then we couldn't always get work done until the other person got it working on their OS. Had we known this issue would arise we would have all picked the same OS to code on so we wouldn't need to keep changing the code based on the OS we were on.

7. SOFTWARE ENGINEERING COMPONENTS

The Python programs were developed using Idle IDE on Windows and Linux. The Java program was developed using Eclipse IDE on MacOS.

Regarding our modeling analysis and design process we implemented UML diagrams into the code using techniques learned while covering coupling, cohesion, and design patterns. We tried to keep coupling to a minimum and cohesion to a maximum. There are instances, in our code, where common coupling took place because we had to revert to using global data to implement our GUI’s using the Tkinter package. Also looking back at our code now we can see areas where stamp coupling took place, because we have different games that refer to the same entire data structure rather than just the parts it needed to access. We didn’t fix this now because of the headache that Tkinter has proven to be when switching even the littlest piece of information.

On the other hand we were able to complete our design with a fairly high level of cohesion. Though there was at least one instance we ran into coincidental cohesion. This took place regarding the different betting styles of the casino games we implemented. Because of the different styles we weren’t able to reuse the code. Informational cohesion can be found throughout our code,

however. Mostly in the form or classes, it is evident that different parts of our code refers to a player class that has different attributes specific to the game being played by the user. Also the different games' GUI's were able to call specific functions, or getters and setters, inside classes in order to carry out computations and logical decisions.

8. UNIT TESTS

A unit test was written for the character based blackjack module using the PyUnit framework. This initially proved quite challenging, as the documentation that comes with the Windows version of Python 3 seems to expect a good understanding of unit testing, which none of the authors had. However, some more basic tutorials were located and eventually a start was made. After that it just became tedious to write a test for every method in every object. One issue encountered was the testing of input and output. There may be an elegant way to handle it, but we ended up making the tests interactive, asking the test conductor to make prescribed inputs (both correct and incorrect) to check for expected responses or error messages. Also the test conductor had to visually check to see that method outputs were as expected.

9. LESSONS LEARNED

The main lesson we learned during this process was to keep the whole project in one language whenever possible. Of course, there will be projects where this isn't possible, but in a project the scale of ours it would have been very feasible had there been an object-oriented language the four of us all knew.

Another major lesson learned was the effectiveness of tools such as UML diagrams and unit tests. UML diagrams were very useful for visualizing how the classes interact with each other within a given program. Unit testing was a very efficient way to assess the quality of our code.

Also, no one in our group had ever used revision control (such as Git) before this project. It was a great experience to work on a project that was constantly changing and to see the revisions as the programs evolved.

10. CONCLUSION

Overall, the members of TeamWu learned many important lessons and techniques over the duration of this project. Though actual work and planned work differed slightly in the end, the process of working together as a team was invaluable. We were glad to have the opportunity to utilize the many new software engineering components learned in CS-680 while designing Casino Games.

11. REFERENCES

[1] Python Documentation <http://docs.python.org/3.2/>

[2] Tkinter Documentation <http://www.pythonware.com/library/tkinter/introduction/hello-again.htm>

12. INDIVIDUAL CONTRIBUTION MATRIX

	Iteration 1				
Tasks	Brady	Seth	Justin	Bill	
User Stories	25%	25%	25%	25%	
Iteration Planning	25%	25%	25%	25%	
UML Class Diagrams	5%	15%	25%	55%	
UML Sequence Diagrams	5%	30%	30%	35%	
Test Cases	5%	5%	5%	85%	
UML Revisions	5%	5%	30%	60%	
Code	25%	25%	25%	25%	
GUI	70%	5%	20%	5%	
PowerPoint	25%	25%	25%	25%	
Report	25%	25%	25%	25%	
Final Presentation	25%	25%	25%	25%	
TOTAL	240	210	260	390	=1100%
out of 1100%	**22%**	**19%**	**24%**	**35%**	