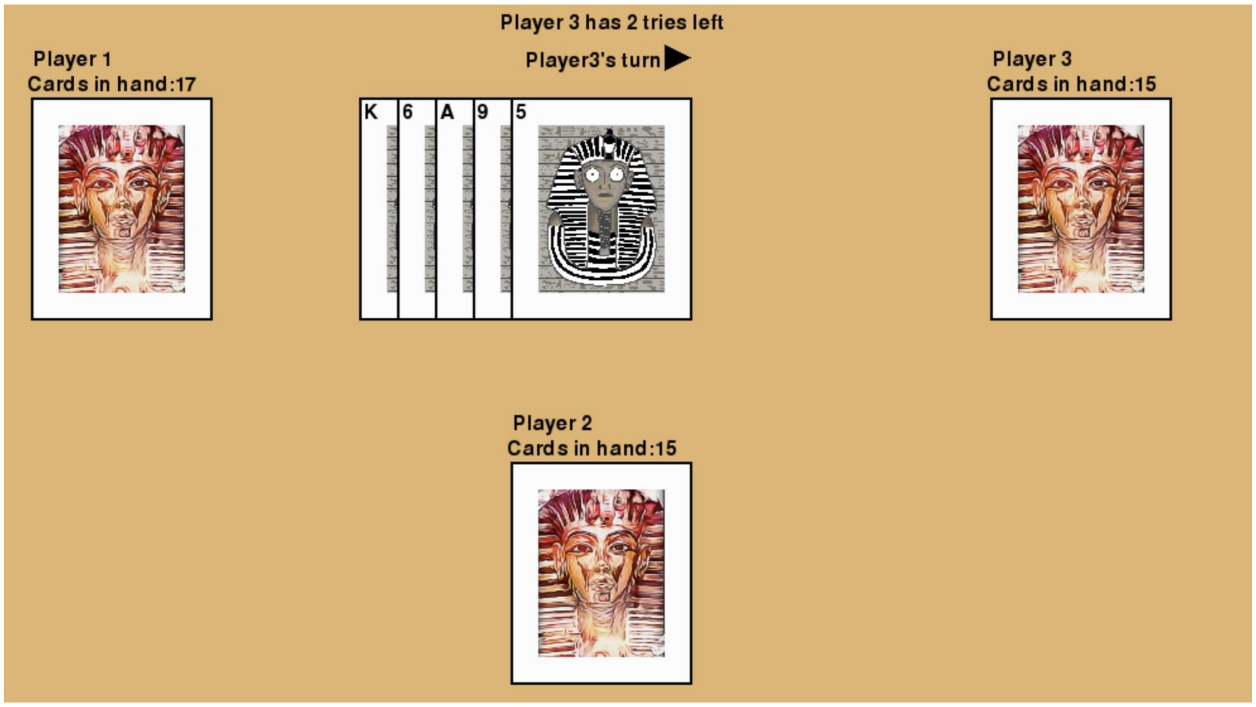
EGYPTIAN SLAP



Final Catapult Report – Team 11
Egyptian Ratscrew

Group Members	Nikhil Nambiar	-	Indus International School Bangalore Karnataka
	Evan Slater	-	Antioch Community High School Antioch

Date Submitted : DD/MM/YY 18/07/19

Project Summary

Our team made an electronic version of the game *Eqyption rat screw*, but the game does go by many other variations of the name, most involving "Egyption rat". At the start of the week we started learning how to use *Python* to code. Half way through the week, the class had learned enough to form teams and start projects.

Our team started by planning out on paper what classes, functions, and inputs we would need and then started putting in the basic setup code for *Python,* followed by the original skeleton code that followed our original plan. Before we started working on any of the functions that we would need for the game we started out by finding out how to deal out cards and play them, using the console to know what is going on instead of using a display. After this we added the last detection we needed from the player, which is slapping. At first getting the inputs was easy and determining who slapped first was as well, but then we ran into a problem

When we tested out the slap class, we were having problems when we connected it up to the win function, where it would run it multiple times, and we saw that the program saw each player slapping more than just one time. This happened to be the biggest roadblock we had in the project since this was the only non logic part of the game. At this point, we realized that our original plan was not going to work, so we had to change our plan.The solution we adopted to fix the slap was that after the first person got rewarded, we programmed it such that when the reward function ran, the player got nothing more.

At this point we decided to split up and work on different parts. Nikhil worked on trying to figure out slap, and Evan working on graphics, challenge, and the logic for the challenge. This was simple since it was straight logic. The only problem was that challenges change how turn order changes. By this time, slap was completed by Nikhil, and he decided to start fixing the problem with the challenges.. After challenge was completed, Evan went and started working on graphics, while Nikhil cleaned up and finalized the rest of the code. After that, we did not run into any major problems other than small bugs.

Finally, we received feedback from other students and teachers and made some finishing touches to the game. We set up instructions for the game, along with a poster to help understand the game visually as well. The game was then debugged and ready to play.

Final Report

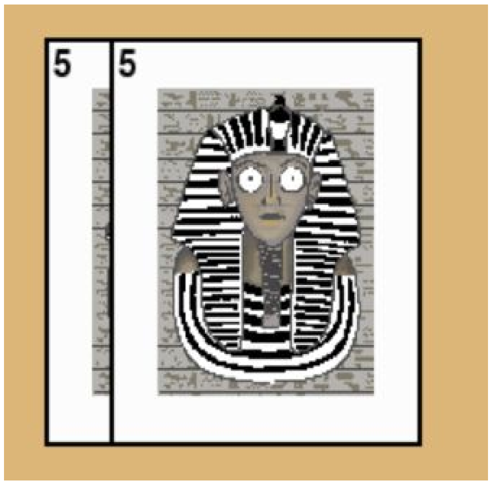
Introduction

The game that we programmed was made using the Python language. Python is an interpreted, high level programming language that was first released in 1991. Python was originally conceptualized by Guido van Rossum in the late 1980s as a member of the National Research Institute of Mathematics and Computer Science. Initially, it was designed as a response to the ABC programming language that was also foregrounded in the Netherlands. Among the main features of Python compared to the ABC language was that Python had exception handling and was targeted for the Amoeba operating system. Interestingly, Python is not named after the snake. It is named after the British TV show Monty Python. Python, like other languages, has gone through a number of versions. The version that we have used is Python 3.7.

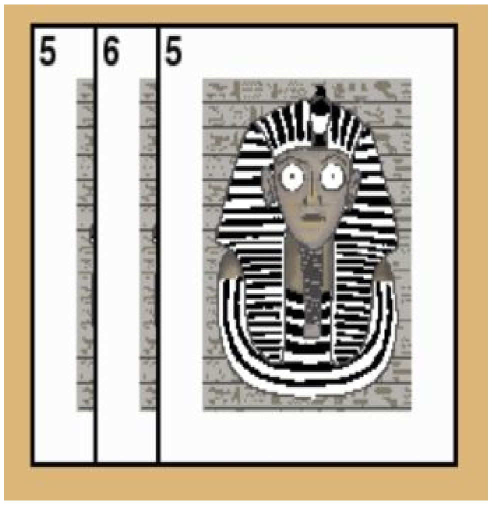
Game Description

We made the classic strategy card game, Egyptian Ratscrew. The winner of the game is the player who wins all the cards in the game. It is a 2-4 player game, that involves a deck of 52 cards randomly distributed amongst the players. The players can't look at their cards as they remain face down in front of them. Players play one card each in order of turns, and when two identical cards appear, any player can 'slap'. Players can also slap when a double with a card of a different value between the two matching cards (like a 5, 8, 5) is played. Whoever slaps, gets all the cards that were stacked on the center pile. Slapping when there is no slapping opportunity is considered a foul, and the player places his next card under the center pile. If any face card or ace is placed down on the center pile, the player who placed the face card 'challenges' the next

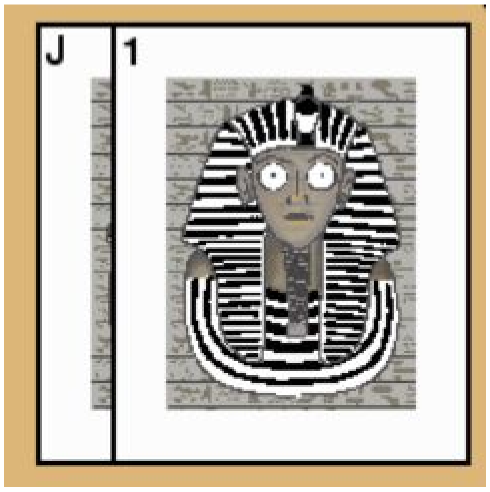
player. The next player has a set amount of tries to place down another face card or an ace to win the challenge (J – 1 try, Q – 2 tries, K – 3 tries, A – 4 tries).
Example of a double slap :



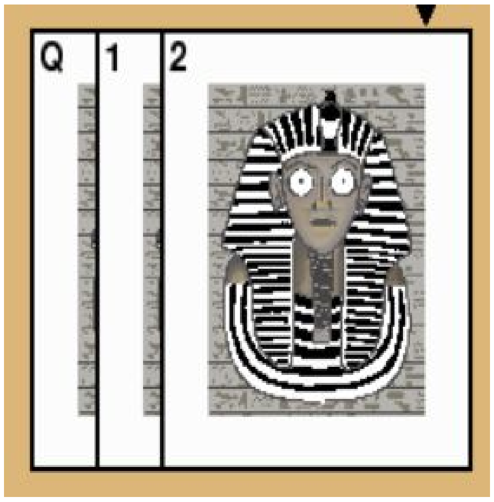
Example of a sandwich slap:



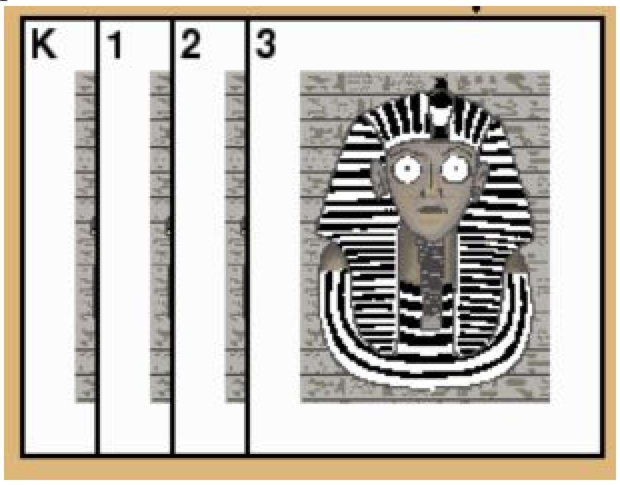
Example of a Jack challenge failed as no face card was placed in one try after the Jack was placed:



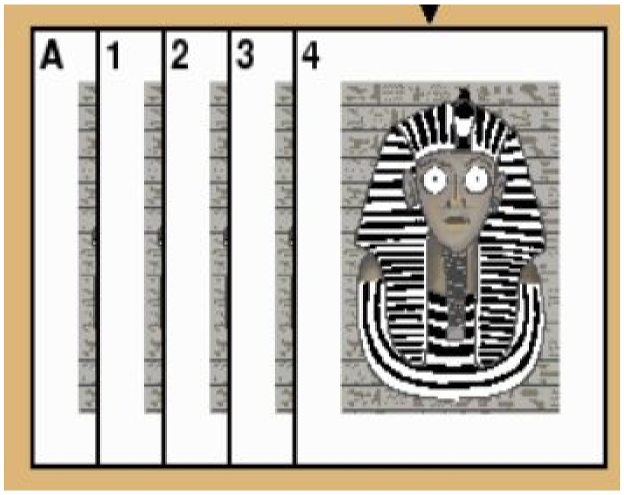
Example of a Queen challenge failed as no face card was placed in two tries after the Queen was placed:



Example of a King challenge failed as no face card was placed in three tries after the King was placed:



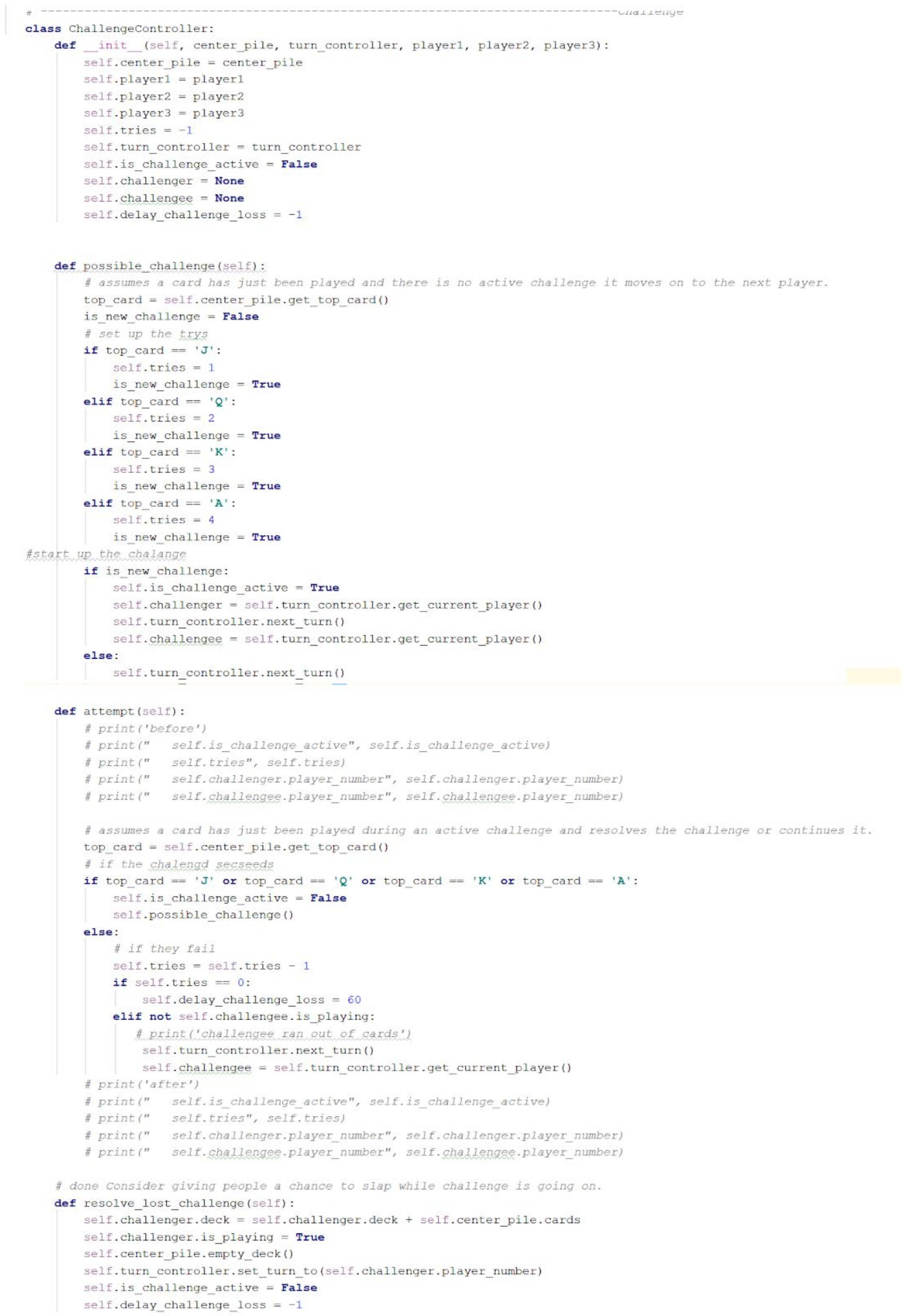
Example of an Ace challenge failed as no face card was placed in four tries after the Ace was placed:



Coding Process

Classes: Player, CenterPile, TurnController, ChallengeController, BoardController
Functions in Player class: place_card, discard_card
Functions in CenterPile class: add_card, bury_card, get_top_card, empty_deck
Functions in TurnController class: set_turn_to, next_turn, get_current_player, get_previous_turn
Functions in ChallengeController class:possible_challenge, attempt, resolve_lost_challenge
Functions in BoardController class:set_up_board, hand_slap, game_over_screen
Functions not in a class: slap, play_card, check_for_game_over, new_game, two_Player

Variables in Player class: deck, is_playing, player_number
Variables in CenterPile class: cards, is_slap_allowed
Variables in TurnController class:player1, player2, player3, previous_turn, current_turn
Variables in ChallengeController class: center_pile, player1, player2, player3, tries, turn_controller, is_challenge_active, challenger, challengee, delay_challenge_loss
Variables in BoardController class:screen, card_image, card_image1, card_image2, card_image3, hand_location, card_location, show_cards, caption_font, temp_storage, who_slapped, slap_sound, card_back_image
Variables in main: clock, screen, card_delay, new_deck, temp_deck, turn_controller, center_pile, challenge_controller, slap_sound, slap_hand, card_image, caption_font, card_back_image, board_controller, is_game_over, has_displayed_game_over, pressed_keys



```
# ----------------------------------------------------------------------------------Challenge
class ChallengeController:
    def __init__(self, center_pile, turn_controller, player1, player2, player3):
        self.center_pile = center_pile
        self.player1 = player1
        self.player2 = player2
        self.player3 = player3
        self.tries = -1
        self.turn_controller = turn_controller
        self.is_challenge_active = False
        self.challenger = None
        self.challengee = None
        self.delay_challenge_loss = -1


    def possible_challenge(self):
        # assumes a card has just been played and there is no active challenge it moves on to the next player.
        top_card = self.center_pile.get_top_card()
        is_new_challenge = False
        # set up the trys
        if top_card == 'J':
            self.tries = 1
            is_new_challenge = True
        elif top_card == 'Q':
            self.tries = 2
            is_new_challenge = True
        elif top_card == 'K':
            self.tries = 3
            is_new_challenge = True
        elif top_card == 'A':
            self.tries = 4
            is_new_challenge = True
#start up the chalange
        if is_new_challenge:
            self.is_challenge_active = True
            self.challenger = self.turn_controller.get_current_player()
            self.turn_controller.next_turn()
            self.challengee = self.turn_controller.get_current_player()
        else:
            self.turn_controller.next_turn()


    def attempt(self):
        # print('before')
        # print("    self.is_challenge_active", self.is_challenge_active)
        # print("    self.tries", self.tries)
        # print("    self.challenger.player_number", self.challenger.player_number)
        # print("    self.challengee.player_number", self.challengee.player_number)

        # assumes a card has just been played during an active challenge and resolves the challenge or continues it.
        top_card = self.center_pile.get_top_card()
        # if the chalengd secseeds
        if top_card == 'J' or top_card == 'Q' or top_card == 'K' or top_card == 'A':
            self.is_challenge_active = False
            self.possible_challenge()
        else:
            # if they fail
            self.tries = self.tries - 1
            if self.tries == 0:
                self.delay_challenge_loss = 60
            elif not self.challengee.is_playing:
               # print('challengee ran out of cards')
                self.turn_controller.next_turn()
                self.challengee = self.turn_controller.get_current_player()
        # print('after')
        # print("    self.is_challenge_active", self.is_challenge_active)
        # print("    self.tries", self.tries)
        # print("    self.challenger.player_number", self.challenger.player_number)
        # print("    self.challengee.player_number", self.challengee.player_number)

    # done Consider giving people a chance to slap while challenge is going on.
    def resolve_lost_challenge(self):
        self.challenger.deck = self.challenger.deck + self.center_pile.cards
        self.challenger.is_playing = True
        self.center_pile.empty_deck()
        self.turn_controller.set_turn_to(self.challenger.player_number)
        self.is_challenge_active = False
        self.delay_challenge_loss = -1
```

Project Reflection

Nikhil's Reflection:
Initially during the planning process, we were struggling to figure out how to start the code. This was after we had decided on the game we were going to make. So, we decided to plan it all out on a sheet of paper and a white board, to organize our flow of thoughts and create an accurate sequence for how the code must be programmed. This helped us work efficiently with a goal set to achieve.
Me, being a beginner in coding and Python, was initially not able to bring much to the table on the technical side of things. Although I did help in organizing the framework and structure of the code. Evan handled most of the code implementation, while I helped with design. After getting a bit more experience with coding, I decided to try out a few code functions on my own. This allowed us to work the code side by side, utilizing time more efficiently, and programming the game quicker. We did run into a few problems later on in the coding process, which had required us to track back on our code and redo a few functions. I could help chip in during this debugging process.
During this debugging process, which was although stressful, I decided to have a cool head and clear thought process to achieve our target. We had an open mindset, understanding the opinions of each other, and evaluating different possible methods to tackle the problems. This definitely helped us achieve our goal within the time frame allotted to us.
As a whole, this experience was surely breathtaking, allowing me to try out something completely new. My interest in the coding and programming world has surely increased, motivating me to continue programming codes in the future.

Evan's Reflection:
At the start of the week, I came into this already knowing what area I wanted to go into, but I wanted to go and see what it would be like taking on a harder challenge. At first when Nikhil and I teamed up we had already decided what we wanted the project to be. This was around the time that I learned that Nikhil was brand new to coding, and then we started to begin planning how we were going to code out the game. We worked well and saw that on paper it looked easy, just very tedious to do.
When we started the code, the project as we predicted things, went on smoothly until we got to the slap feature. At this point this was a hard road block that we had. This one problem took a larger chunk than all the other parts. We needed to get help to get this part working, in a way that would not mess up any other part of the code. But we got through it.
After we got past that, we decided to split up so that no one person was doing all the work and we could work faster as a team. We split it up so that I was working on the graphics and challenge logic, while Nikhil was working on turn order and getting the two parts ready to merge.
With Nikhil being new to coding he was able to bring up ideas to fix problems that people who have coded already would forget or not think of. This and his mindset to be open to something brand new to him made it so that even though he did not know how to code that well, he was an important part of the team. Along with this the mind set that I came in with was one of not letting

my group make decisions without thinking and making sure that we did not stray too far from what we were supposed to do. The skill that I brought with me let me help my team do more complicated things of code and it not have problems because of errors that are commonly missed. With the ambition of someone new and the knowledge of someone who knew what they were doing we were able to get through the code with little to few problems.