**NSN NUMBER: 0135936994**

**Blackjack With Betting**

**Luka Jeremic**

**Develop an Outcome L2**

Evidence of Learning

**Links to/for Completed Projects:**

* [Trello Board Link (Project Planning Throughout)](https://trello.com/b/HONpgoEW/blackjack-mags-dgt)
* [Drive Link to completed Python file](https://drive.google.com/drive/u/1/folders/1hDog4kGmvMZoVKZbckMYwC3UvP01SnPo)

**Time Frame:** 10 Weeks

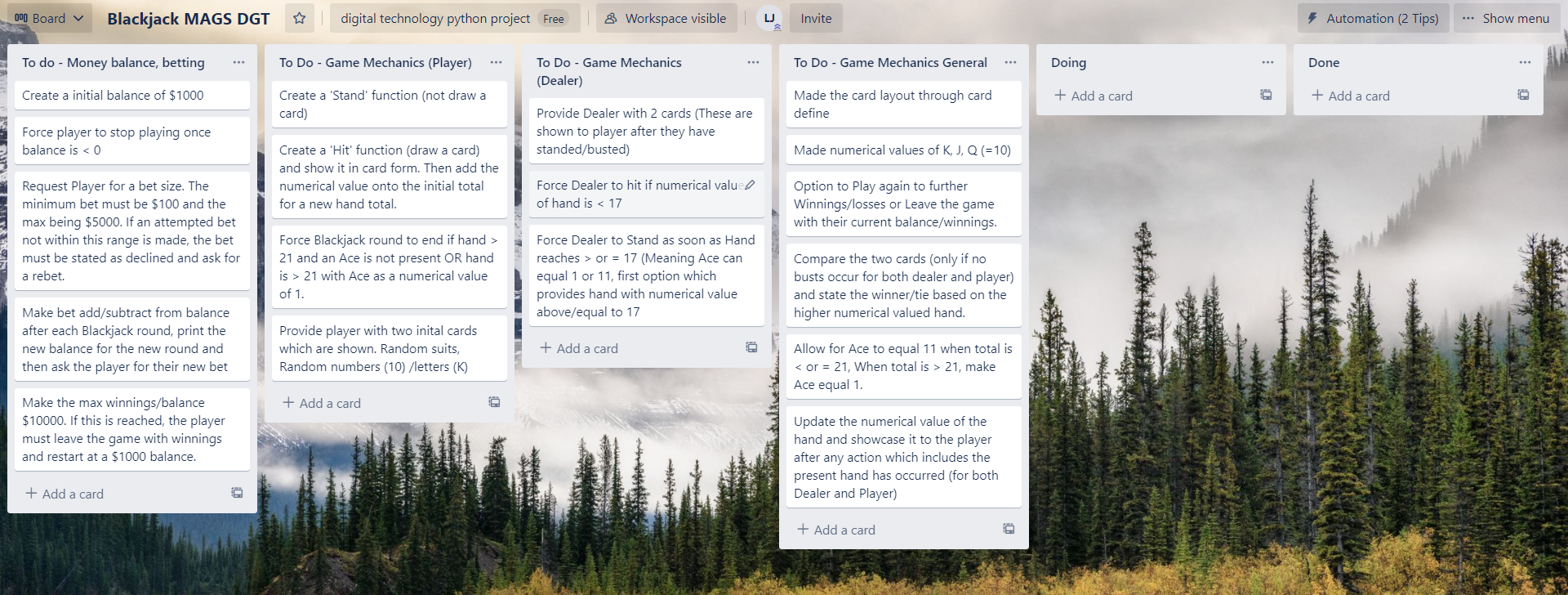
**Milestones:** Establishing and achieving Weekly Goals (decided in my mind)

**Workers:** Home PC and Laptop at school. The code file was shared between both workers through updating a drive folder with the new code after each session.

**Project Management: Setup - Tools**

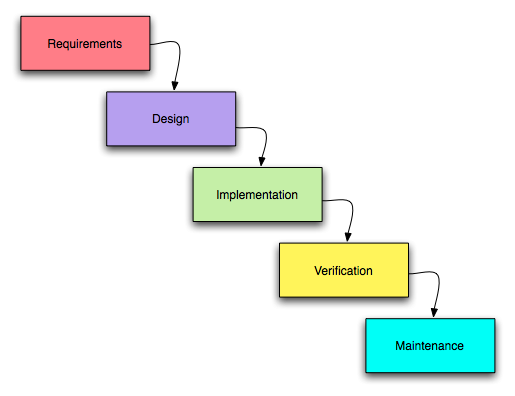
* **Record evidence that you have set up project management tools to manage the development of your outcome.**
* **Explain why you believe that these tools are appropriate for the development of your outcome.**

**Trello Board:**

For my project management, I used a trello board. The trello board allowed me to break down my project into multiple, smaller tasks which I can complete in stages. These stages included: To Do: Money balance/betting, To Do: Game Mechanics (Player Specific), To Do: Game Mechanics (Dealer/Computer specific), To Do: Game Mechanics (General). I also created a ‘doing’ tab which allowed me to keep track of what parts of my game I was coding at each time. Furthermore, these game sections were broken down further into specific jobs which would be needed for my BlackJack code. These stages and sections completed my original setup of my Trello board shown on the right. 

The Trello board also allowed me to easily locate problems due to tasks being split up, and made it easier to improve my code using sections instead of one large code. This was crucial in the success of my BlackJack game as this would be my first year coding and I needed as much support as possible in creating such a game.

**Coding Process:**

I also used an online coding process in order to maximise achievement of my code. This starts with the Requirements. For each task, I would understand the requirements and overall what needed to be achieved. Following this, I would design pieces of code which completed these requirements. I would then implement it into my main code and test for errors, then double check it if it completes the requirements and was designed to the best of my ability. This would complete the verification section of the coding process and lead me onto my final step which is maintenance. For this section, I would regularly check up on the code and specifically these sections. I made sure that changes in other parts of the code did not affect the designed code in question and that It was still meeting the requirements made in the first section. This overall process allowed me to create a very reliable code despite the numerous functions which would be present at one time.

**Project Management: Setup - Techniques:**

* **Record evidence that you have chosen project management techniques to manage the development of your outcome.**
* **Explain why you believe that these techniques are appropriate for the development of your outcome.**

**Trello Board Techniques:**

**History:**

A very useful feature I found on the Trello board was its history. This meant that my activity would be saved and gave me confidence in completing many tasks. This meant I was not worried a task would accidentally ‘disappear’ and I was able to forget about it and focus on other tasks all thanks to the history feature.

**Archive:**

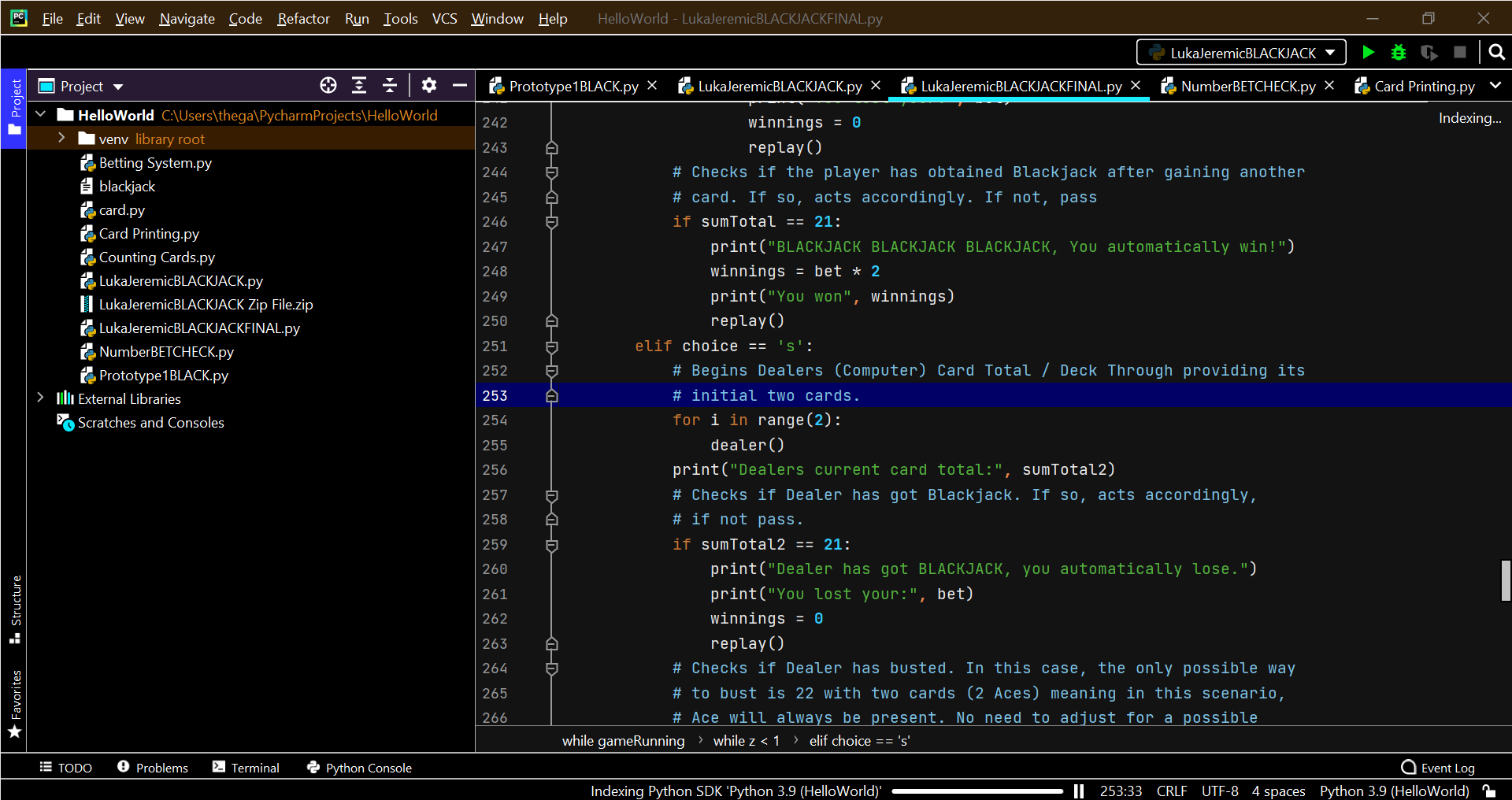
While planning on my Trello Board, I used the ‘Archive’ technique to remove tasks which I have already completed. The key benefit of this feature is that it would be saved into a separate folder. This means that if needed, the task would be able to be retrieved. This made it incredibly easy to remove tasks as I knew I could always retrieve them if needed. This was incredibly useful when determining if tasks were ‘necessary’ or not. If I found it to not be necessary and very tedious to create, I would archive it so that it was no longer a task. However, I had confidence that if for some reason, it ended up being necessary or I had extra time, I could bring back the task and continue working on it.

**Watch:**

The watch feature was also very useful. It allowed me to be able to tag specific tasks that needed special attention. This was either due to the difficulty of the task and needing to do it later, or simply needing to stop mid task due to other reasons, and also needing to do it later. It allowed me to realise what tasks I still have not completed, but have attempted. This highlighted possible difficulties that may need to be addressed later.

**Project Management: Setup -**

* **Record evidence that you have set up/ chosen version control tools/techniques to manage the development of your outcome.**
* **Explain why you believe that these techniques are appropriate for the development of your outcome.**



**BlackJack Versions:**

**T**he image on the left shows three different versions of my Blackjack code. These being: Prototype1BLACK, LukaJeremicBlackjack, and LukaJeremicBlackjackFINAL.

Prototype1Black was a very basic version of my BlackJack code. It is designed to test all my different game variables and verify they all work together to get the basis of the game working.

LukaJeremicBlackjack was my first proper Blackjack code. This featured better dialogue for the user, further testing and refinement from Prototype1 and comments on parts of the code.

Finally, there was my final code LukaJeremicBlackjackFINAL. This code has been thoroughly tested by family and friends and has overtime been fixed for slight errors, lots more comments were added and the dialogue has been refined. The key difference with this code when compared to other two codes was that the code was PEP8 verified and correct Syntax for a final, fully complete and correct BlackJack code.

**Decomposing Outcome:**

In order to increase efficiency and probable success, I broke down my outcome into parts which will be separately coded. This means that If something is needed to be added, I can code it into the smaller projects making me less confused with the many variables. It is also beneficial to do this so that if something were to not function as intended, then I can quickly deduct the issue as all codes are separated into sections and continue creating my Blackjack.

**These sections are:**

* Dealing Cards (Printing Card Layout + Randomising Cards)
* Betting system (Provide $1000, Minus or Plus Winnings)
* Repeatedly ask Hit or Stand when Suitable
* Function of Hit
* Function of Stand
* Replay/Redeal (Restarts the BlackJack round for another game)
* Ace checker. (Checks the players hand for an Ace)

I decided to code these sections in this order:

1. Dealing Cards
2. Function of Hit
3. Function of Stand
4. Repeatedly ask for Hit or Stand when Suitable
5. Ace Checker
6. Replay/Redeal
7. Betting System

I will code the dealing cards first as this is the core of my game. Without this, there will most definitely not be any part of my BlackJack and would be a very bad experience for the user.

Secondly, I will be coding the Hit function. The Hit and Stand functions are the two most important commands in Blackjack. It is a core game element and for my game to be considered as Blackjack, these must be included. I am coding ‘Hit’ first as this is the more complicated function when compared to ‘stand.’ This is due to the gaining of a card creating more plausible outcomes, and also adding to the players current numerical card total.

Third is the Stand function. As mentioned earlier, this is a core element of my Blackjack which is why I am coding it before other sections to ensure that it will always be working. This follows the Hit function as there is less which occurs when the player Stands and therefore it is easier to have working.

Next is repeatedly asking the user for these two functions. This is an important function as it is also a core part of the Blackjack game. This is only suitable when: The player's card number is <21. If the player has gone over 21 or has chosen the ‘Stand’ function previously, then this question must stop.

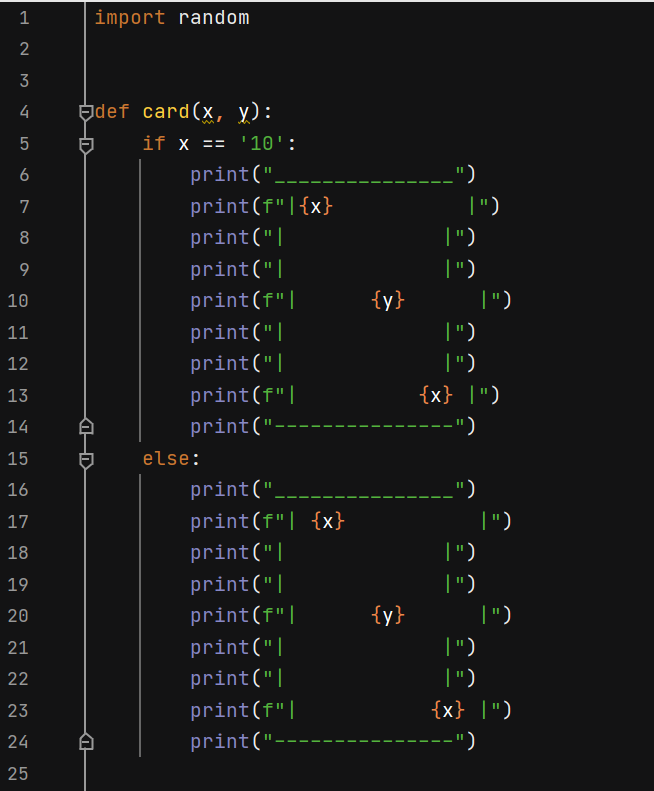
After this will be my Ace Checker. The purpose of this is to check both the player and dealer's hand (current cards) if there are any Aces. This is because the ACE in Blackjack has two possible values. These being 1 or 11. I decided that to do this, I would have the ACE default as a value of 11. Then, if the player goes over 21, the code will use the Ace Checker to detect if an Ace is present. If so, the Ace will convert to a value of 1 and the player will no longer be over 21 and continue playing.

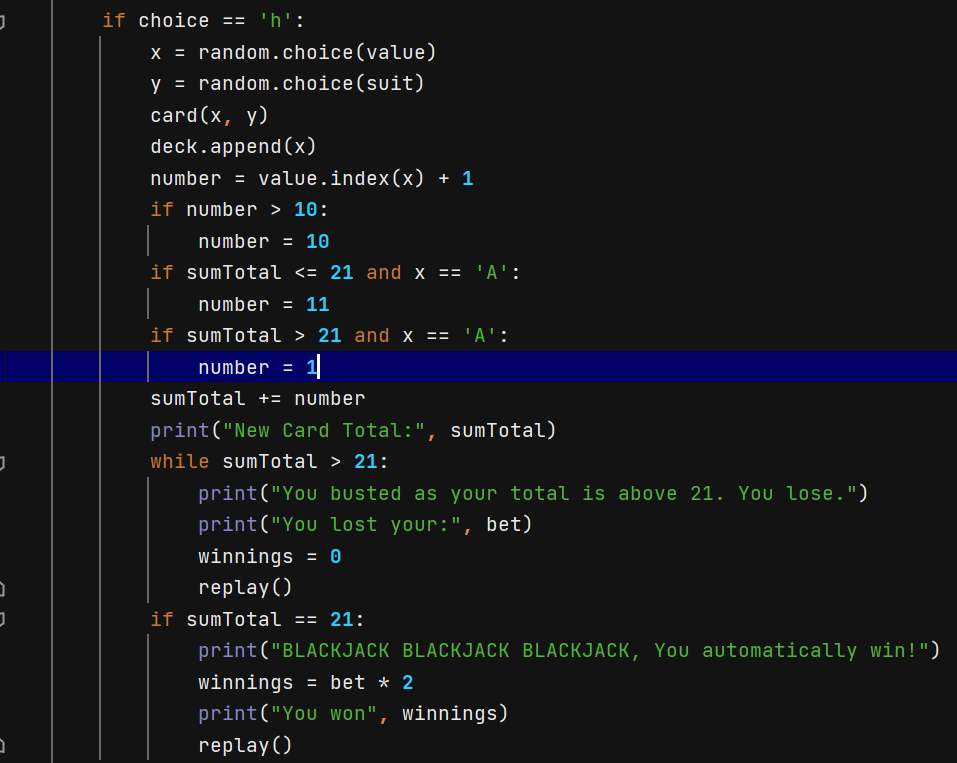
Replay/Redeal is the 6th piece of code I will create. The purpose of this is to ask the player if they would like to play again. If so, the code must rerun itself and all values must reset. If not, then simply display an ending statement of the players winnings, etc.

Finally is the betting system. This is last and the least of my priorities as it is not actually part of my specific BlackJack game. My BlackJack will work completely fine without it. This is simply a cool add on to keep the player interested in playing.

**Trialling Components:**

**Dealing Cards Component:**

A core component of my game was the printing of Cards. I wanted to have a component that would randomly print a Card onto the player's terminal. It would print a random suit (e.g spades) and a random card value (e.--g King, 10), and the border forming the shape of a traditional playing card. On the left image is the original concept but after some trialing, I realised that the spacing of the card becomes incorrect when x = 10. This is because unlike all other symbols, the 10 symbol takes up two spaces. With this in mind, I made a slight change so that if x = 10, then there would be a specific spacing. This meant that I now had a complete first component of my BlackJack game.

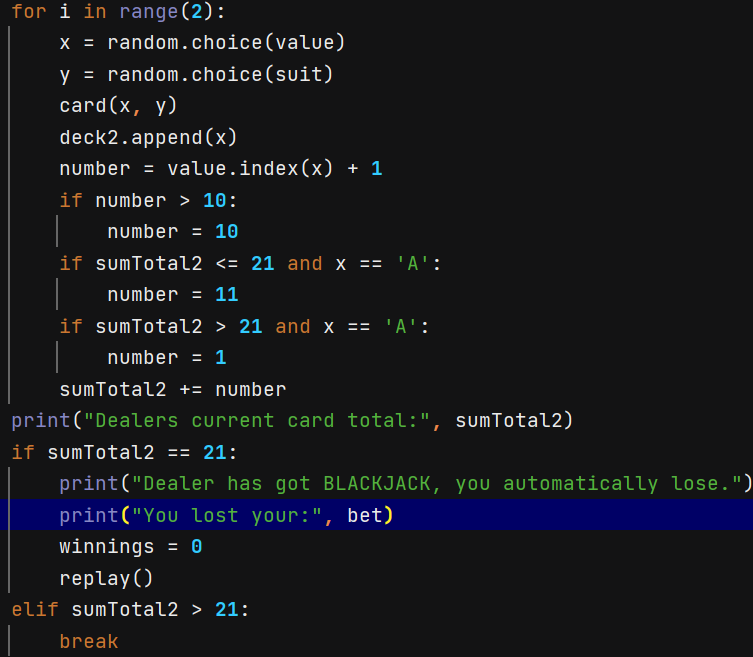


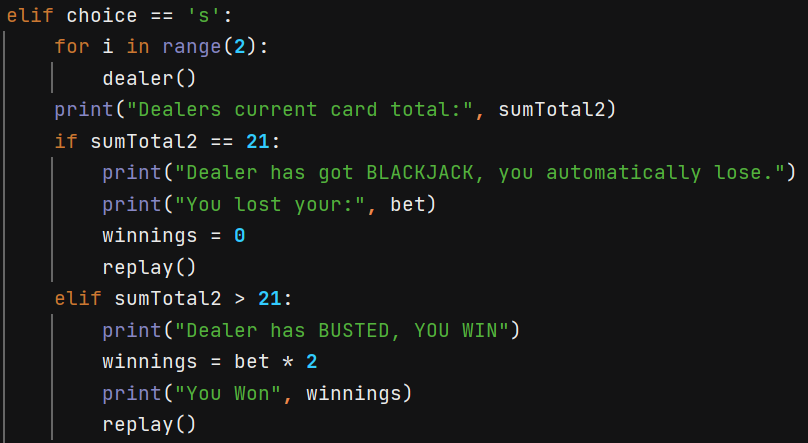
**Hit Function Component:**

Another core component of my Blackjack game is the ‘hit’ option. ‘Hit’ (also known as obtaining another card), must provide the user with an additional card (Through printing it onto the terminal from the card component) and add the numerical value of the card onto the user's hand numerical total. Furthermore, It must check if the user has ‘busted’ (Gone over a numerical hand value of 21) and if so, declare the round to be over as a loss. The screenshot on the right showcases the final product of this component. Through trialing, there was only one change needed where a possible sum total of 21 was not accounted for. This was a simple addition and never failed me so I decided to continue with it.

**Stand Function Component:**

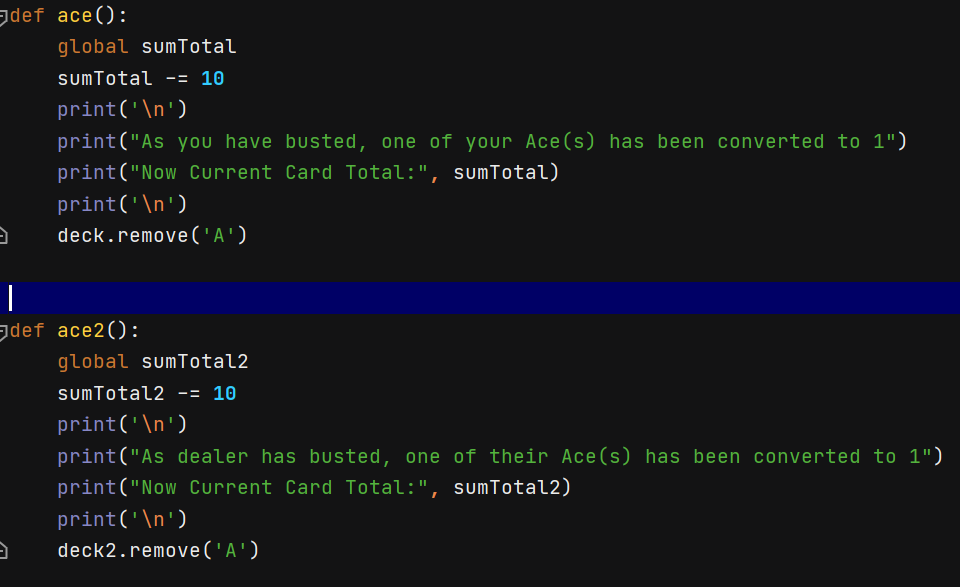
Standing is a very vital part of the Blackjack game but quite simple to complete. If this function is chosen by the user, it must simply declare the user's numerical hand total and proceed the round to the computer generated hand, also known as the Computer Player component.

**Computer Player Component:**

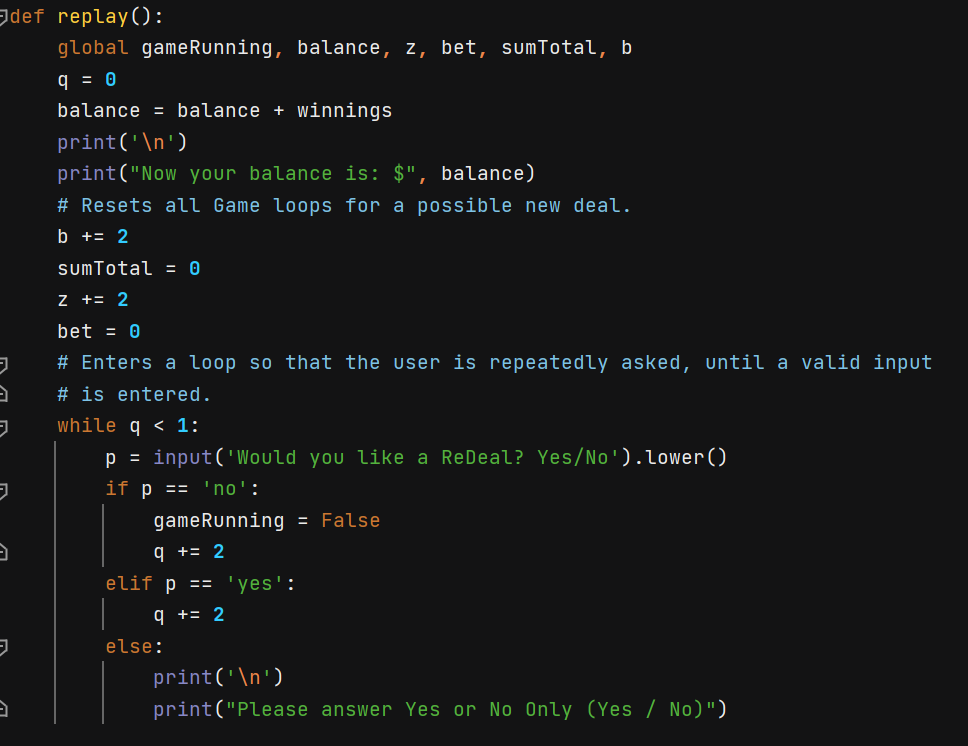
This is the most crucial part of my Blackjack game as it is centered around the user's opponent. The code must provide two random cards with the card component, accumulate the numerical value and follow this rule: If the numerical value of the hand is greater than 16, the computer must ‘stand’ and move onto the endgame phase component. if the numerical value of the hand is ever 16 or below, then the computer must ‘hit’, adding to the numerical value of their hand and repeating the rule. Also like the ‘hit’ component, if the computer's numerical value of their hand ever exceeds 21, then they have ‘busted’ and they must be declared as the loser. On the right is the original prototype of this component. Through trialling, I realised that the code worked well, but could be made simpler and dialogue could be more clear for the user. Due to this, changes were made into the final code design of the Computer Player component:

The cards have now been defined for a simpler code as well as dialogue + other values have been added if a BUST were to occur.

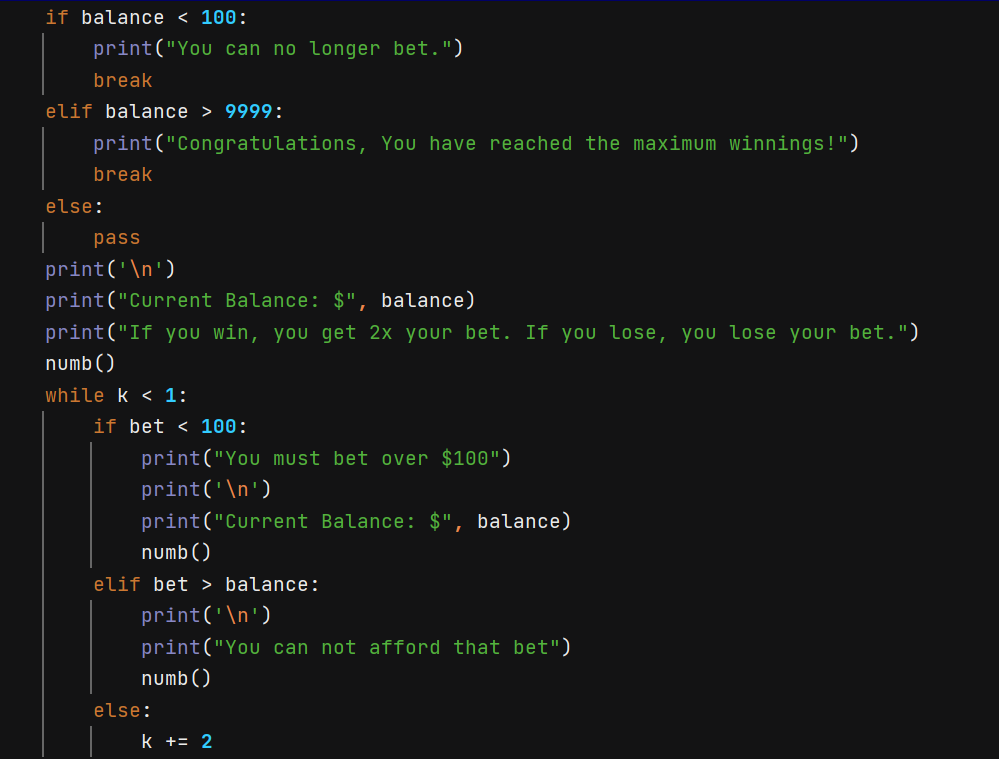
**Ace Check Component:**

Likely the most mechanically difficult procedure is checking if the computer or player has an Ace. This is important as an Ace in official Blackjack can have a numerical value of 1 or 11 based upon the user's choice. I decided that the most effective way to do this would be to have the numerical value of the Ace equal 11 at default. The computer/user plays as normal until they have busted. As they have busted, logically they would like to convert their 11 to a numerical value of 1 as it would prevent them from losing. The screenshot on the left showcases both components complete. After understanding how to complete the procedure, it was not very difficult to make it a reality. Through trialing, I experienced no problems and I have kept the component the same ever since its first creation.

**Replay/Redeal Component:**

This component asks the user if they would like to play again or take their winnings and stop playing. The purpose of this component is to reset ALL values and loops so that if there is another round, everything will be reset to default. This itself was not very difficult to achieve and only needed one iteration for a working design.

I understood that this must be defined as there are many possible outcomes in Blackjack meaning many areas where this replay question would be needed. I also used a loop to ensure that the user inputs an answer that can be understood. This was trialled several times and worked consecutively, serving its purpose of resetting the game loop.

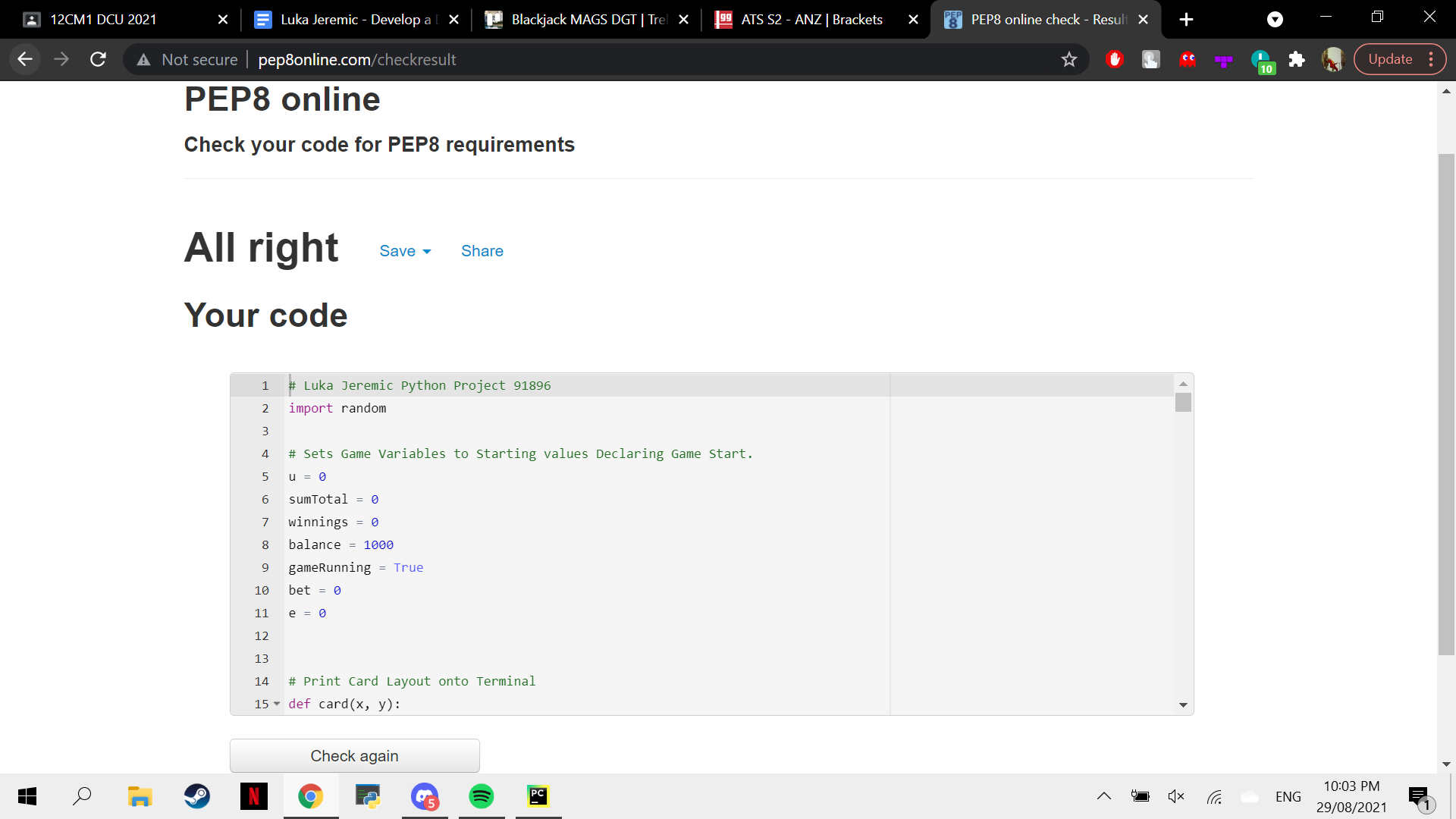
**Betting Component:**

When the idea of my BlackJack was introduced, I received feedback of a plausible betting system. This added more sense of ‘winning’ and overall made the game more competitive and interesting. Thus, I needed a component that would track the balance of the user whilst they are playing, and have a minimum balance of which you can bet and play. On the right is the final version of the component. This was quite simple and did not need many iterations. There are just many ‘if statements’ to cover all possible outcomes to confirm a working and enjoyable game/experience for my user. All possible outcomes were trialled and confirmed to be accounted for.

**Testing Outcome:**

[Video of Debugging FINAL CODE](https://drive.google.com/drive/u/1/folders/1SzIhDtrQfZl3v-n39nHq7oAJ9Gbs05yT)

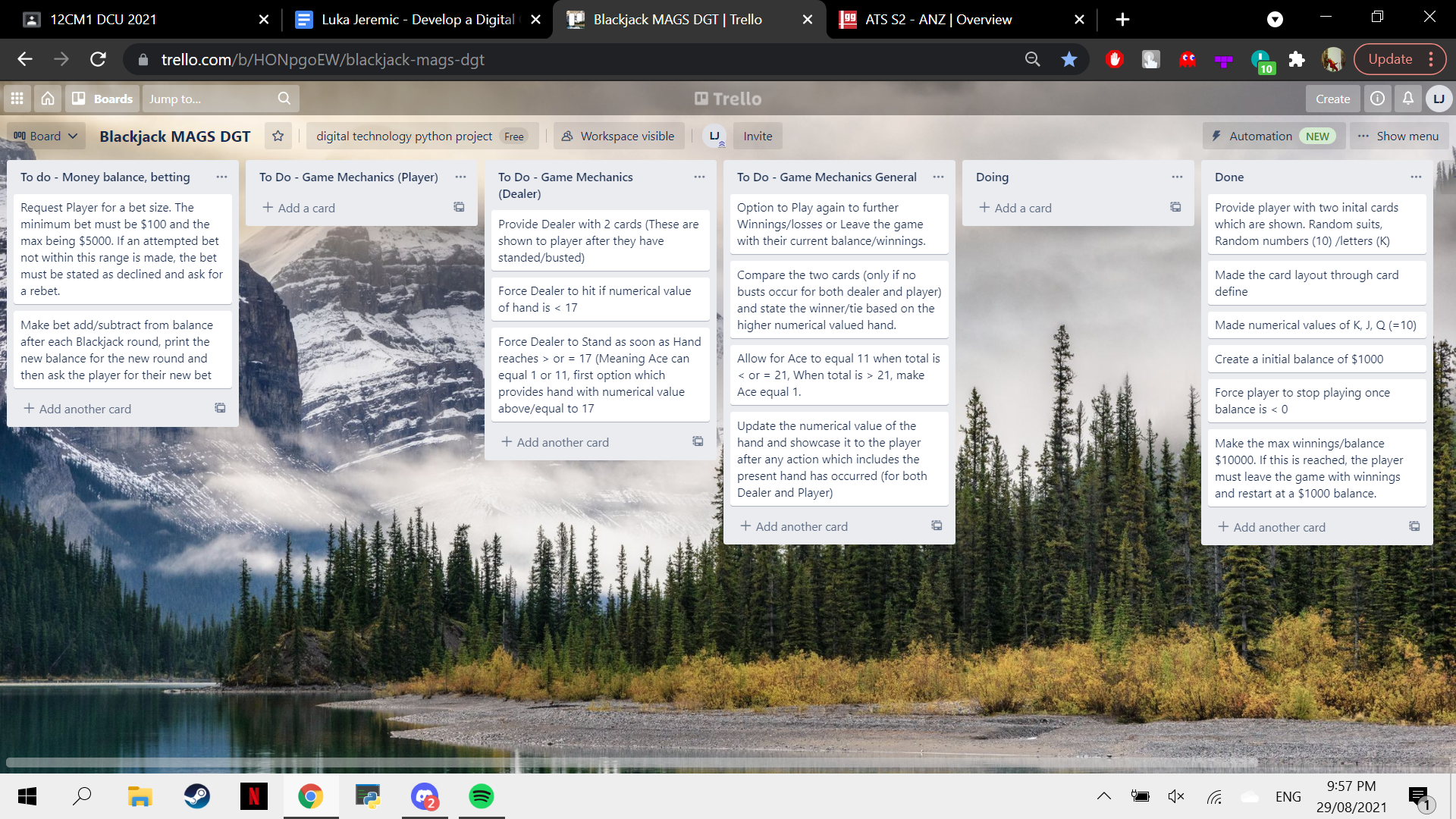
PEP8 Validation: PASS



**Peer-Testing:** “Very enjoyable and easy game to play. The betting element made the BlackJack more intense and competitive when comparing who can reach the highest winnings.” - Arnav (Friend).

**Peer-Testing 2:** “The game itself was very well executed. However, certain commands such as the Hit or Stand should be made easier to answer due to its repetitive nature. A simple H or S symbol answer will make the game more fast paced and enjoyable.” - Max (Dad).

Taking this feedback into account, I made some very small changes and finally confirmed the completion of my BlackJack game.

**Project Management Applied:**

This was my trello board ¾ of the way through my BlackJack code. As shown, There have been sections of the trello board that have been completed and moved to the ‘Done’ section. However, It must be understood that the ‘Done’ section is only recently finished components. These are not 100% confirmed to always be working and should be looked at again soon to double check they are working correctly. However, if that component has 100% been confirmed as working, then the component would be archived and not be part of the trello board as it does not need to be worked on anymore. This showcases the application of the Project Management mentioned at the beginning of my digital outcome making process.

**Version Control:**

**Multiple Copies:**

During the creation of my digital outcome, I made sure to save multiple copies of the code (Typically 3) after each change. This is due to the complicated nature of my digital outcome. I decided that the best way to create my digital outcome was to attempt to implement each component to the best of my ability. However, if it were to fail for whatever reason, as I have multiple copies saved, I can fall back onto the previously working code and reattempt in addressing the implementation of the component. This overall made the coding process of my digital outcome very easy as I always knew I had a working code to fall back on.

**Implications:**

**Privacy:**

An Implication known as ‘Privacy’ was taken into account when I was creating my digital outcome. The ‘Privacy Implications’ is where user’s data which has been provided stays confidential and is not used in any way it was not desired to/agreed upon.

In my digital outcome (BlackJack game), personal user data could be obtained through the question of “What is your name?” which is used to make each round unique to each player.

My program allows for numbers to be used (only if it is not the first character), and any length of name. It also allows for names that do not need to be their legitimate or user real names. Instead it could be fictional usernames, codes, initials, etc. There is also no storing of the data in order to keep the user’s data confidential. As soon as the game is over, the code will reset and the user’s data will vanish along with all game records so that it will be forever confidential and therefore ‘private’.

All the features of the code listed above are important to safeguard privacy and to build trust with the user. This may apply not just for the digital outcome of the Blackjack game but also for any possible future projects.

**Social:**

When creating my digital outcome, ‘Social’ implications were of a big importance. ‘Social implications’ is where the user should not feel an effect from being in a specific social grouping. Social groupings are considered as: Ethnicity, Age, Gender, Religious Beliefs, Geographic location, socio-economic status, etc. It is important that a digital outcome appeals to all social groupings, where possible as this prevents the misrepresenting of a social group or stereotyping people.

My digital outcome has taken ‘social implications’ into account through not having anything that could be seen as offensive to social groupings. Although my code has lots of dialogue, these are all BlackJack/Game based and nothing in my code could be seen as having an effect on social groupings. Consequently, I believe my digital outcome has met the social needs of users and also meets the social norms of modern society.

**Legal:**

The “Legal implications” cover three main aspects. The Privacy act, the Copyright act, and the Harmful Digital Communications laws. The Privacy act is very similar to the Privacy implications. The Copyright Act highlights what is determined as ‘copyright’ or original creators of things like the brand of cards used in the game and legal implications. Finally, the ‘Harmful Digital Communications laws’ which highlights the possible harmful statements that could be said in a digital outcome.

In my digital outcome, The Privacy Act had already been mostly addressed through the Privacy Implications. The Digital Communications laws were also addressed as my code was very broad and generalised. None of the used dialogue could be interpreted in a harmful way which could violate this law.

However, The Copyright Act however is something that I had to be very conscious of. As my BlackJack game was centered around playing cards, I had to make sure that I would not get a copyright complaint through the use of somebody else’s playing cards such as PNG. This led me to the decision to create my own layout of playing cards. This was a simple print function and was highlighted in the ‘dealing cards component’. This prevents any possible copyright infringements by Card Brands like PNG.

To conclude, I believe all of the acts/laws were taken into high consideration and my digital outcome meets all the legal implications.

**End User:**

The End User implication is the consideration of the user itself when using your digital outcome. It is to ensure that the digital outcome has what the user needs. Some of these being: Will it be easy to use?Is it in language suitable for a user to read?Will it appeal to the right demographic?.

A significant consideration was given to the experience the end user will have when using my digital outcome with full intention to have the process incredibly simple to use. For example, questions are always provided with possible answers which the user can pick from. Also a description of what has just happened (e.g player has hit for a new card total of:). Furthermore a description of why the player had just won/lost. Finally, I made sure that the language used was suitable for the user through having a mini ‘terms’ section at the start of the Blackjack game. This clarified Blackjack ‘terms’ and language which could not be initially clear to the user. This implication was thought of highly when creating my digital outcome and I believe that the finished product still follows this implication creating an enjoyable experience for the user.

**Evaluation:**

Although I believe that I managed to code my BlackJack to the best of my current ability, I think that there are some areas where I could have realised its importance and spent more time improving on.

Firstly, while coding my game, I believed that if the code worked, then it was coded well and in the correct way. However, I have come to realise that even though a code may have worked, it may have not been efficiently made. The reason for this was that the structure and the way I have coded my game was not very optimal. It used far too many loops and was over complicated for no particular reason. If I were able to go back and work on my code, I would definitely take a step back and consider more thoroughly, how I will code my whole game differently in order to not only increase efficiency, but also reduce unnecessary over-complication.

Another area where I believe I could have done better was my trialing. The BlackJack game is very ‘luck’ based, therefore, it was very difficult to test certain situations and determine if they were functional. Examples of these situations included ‘Double Aces’, ‘Double Kings’, or even gaining a ‘10 of a Suit’. Some situations were extremely rare that they would hardly ever happen during my testing. I ran the code numerous times until I had that specific, rare situation occur once. I then used this one situation to determine if that section of code was deemed functional.

Looking back, I have realised that this is a **huge** mistake. This is because I have not accounted for all scenarios. This being the possible change of side values/variables which can slightly alter the situation. I have learnt that components must be trialed at least 5 times (in my opinion) to then determine their functionality.

I concluded that I must be more patient in my testing and test all scenarios to confirm all possibilities. This will result in a positive outcome for the user irrespective of the situation and how rare they can occur.