FIT3077 Sprint 3

Team InfiTech

Shannon Wallis, Ziheng Liao, Harshath Muruganantham

Monash University

FIT3077 - Software Architecture

18/05/2024

Table of Contents

Introduction

This sprint, Sprint 3, involved our Team (Infi-tech) combining our initial individual prototypes developed in Sprint 2 together in the most optimal manner we saw possible.
This process began by initially evaluating each of Shannon, Harshath and Ziheng's submitted prototype against a range of quality characteristics and associated metrics that we had pre-defined.
The process of evaluation involved the prototype creator simply demonstrating to the other 2 reviewers in our team, how their executable worked and allowing us to view their code and UML/Sequence diagram designs.

This process provided us with an unbiased evaluation opportunity that ultimately facilitated the process of choosing how to design our combined and complete development executable.

We present the final solution in a video format and include the source code in our final submission. This is supported by the documents we created to initially design it, including the updated class diagram.

To watch the video demonstration of our finished product, click this link to youtube: https://youtu.be/SQewFzvrfos

The executable is the complete game. To run the game, it must be executed on a Windows based operating system. If for some reason, you are unable to run it, you can create an executable by running the command:

pyinstaller --onefile --noconsole Project/src/StartPage.py

You will then need to drag the executable out from the *dist* folder into the Project directory which is 1 level outside of the *dist* folder. Instructions are also on the READme file.

The UML Class Diagram is included at the end of this document. A link to view it is here: https://lucid.app/lucidchart/e54cb2f1-75ca-4aa0-8f6d-ab7f434f7cba/edit?viewport_loc=-5329%2C-1823%2C17989%2C9221%2CMjxBZMjm_W5E&invitationId=inv_20ffc2b8-cee7-40dd-93f8-ebc46c4b801b

Quality Characteristics and metrics for evaluation

Functional Completeness -

The general definition for functional completeness from the international organisation for standardisation is *the degree to which the set of functions covers all the specified tasks and intended users' objectives.*

Our definition of functional completeness is when a given prototype fulfils the stated objective it agreed to upon its creation.

There is no explicit metric that could possibly measure functional completeness without being overly broad and thus, redundant in its purpose. For this reason, we have defined 6 sets of different metrics to be applied when measuring this quality characteristic. The metric to be used depends on the stated objective of the prototype.

Functional Completeness Metric 1: Consists of *"setting up the initial game board (including randomised positioning of chit cards)".*

- Upon game start up/initialisation, 4 distinct caves cards are positioned and visually evident around the volcano 'ring'
- Upon game start up/initialisation, 24 animal's (volcano card steps) are featured on the board. Each animal is positioned such that 1 is not overlapping another
- Upon game start up/initialisation, 4 player tokens reside on top of each of the cave cards. Each player token can be visually distinguished from another
- Upon game start up/initialisation, 16 chit-cards are visually evident. Each card is separated from another so that they are not overlapping. The card's have the same dimensions.

Functional Completeness Metric 2: The game has the functionality of *"flipping of dragon (chit) cards".*

- Upon game start up/initialisation, each of the 16 placed chit-cards on the electronic game board are identical, visually to a player.
- After game start up/initialisation and upon a player's mouse selection at the position of a "chit card", the card changes its look to reveal an animal image instead of the identical "cover" that existed on initialisation.

Functional Completeness Metric 3: The game allows the *"movement of dragon tokens based on their current position as well as the last flipped dragon card"*

- Upon correct selection of a chit-card by a player (ie: the case where a player chooses a chit-card where the animal underneath is the same as the volcano card step animal it resides on), its corresponding player token advances clock-wise forward.
- Upon incorrect selection of a chit-card by a player (ie: the case where a player chooses a chit-card where the animal underneath is NOT the same as the volcano card step animal it resides on), its corresponding player token does not move forward nor back around the 'ring' of volcano cards.
- Upon selection of a pirate-dragon chit-card by a player, its corresponding player token moves counterclockwise around the volcano card ring.

Functional Completeness Metric 4: The game allows the functionality *of "changing of turn to the next player"*

- Upon incorrect chit-card selection by a player (if the selected chit-card is not a pirate type, or an animal type that the player currently sits on), the next chit-card selection will move the next player in line's token rather than their own (changing player turn)

Functional Completeness Metric 5: The game allows the concept *of "winning the game"*

- Upon selecting a chit-card that could progress a player the correct number of steps back into their initial starting cave, the player is deemed a "winner", and cannot proceed to make more turns in the remainder of the game

Functional Completeness Metric 6: Objective of *"Deciding on player turn order for the game duration"*

- When the game executable loads and opens so a player can see it, the player's are able to decide among themselves to choose who will take first turn, second, third and last turn after mutual agreement through verbal discussion.
- As a player chooses their turn position (1st, 2nd, 3rd or 4th), they are placed in their respective cave. The first placed player is placed in the top cave on the screen. The next player is placed in the clockwise cave to the top and so on.

Functional Correctness:

The general definition for functional completeness from the international organisation for standardisation is the *degree to which a product or system provides accurate results when used by intended users.*

The group's definition for "Functional Correctness" is such that a game can only be considered "Functionally Correct" if the software is able to achieve the exact intended effect of performing the game's function as specified in the rule book. It goes beyond functional correctness in that the feature must exhibit the properties/characteristics of a standard Fiery Dragon's game, rather than just 'working' as determined by functional completeness.

Functional Correctness Metric 1: The ChitCards are randomly generated per game

- No 2 boards will be the same across consecutive generated boards.
- In the case where 2 boards generated are the same, to ensure it wasn't a coincidence, we will generate the board 4 times and compare whether the position of the chit cards are the same (to deem the chit-card positioning 'not random')

Functional Correctness Metric 2: Flipping ChitCards

- The 16 chit-cards are positioned in a uniform fashion, be it grid or otherwise, to ensure that there is no assistance to players in memorising
- The chit-card is flipped 'in-place', meaning its location on the screen when revealing an animal is the same as when it is hidden to a player
- The same Chit-Card flipped across the duration of the game, will reveal the same animal underneath every time.
- The chit-card is 'hidden' to a player when the player's turn is over. That is, upon the change of player turn, the chit-cards are indistinguishable from each other.

Functional Correctness Metric 3: The game allows the movement of dragon tokens based on their current position as well as the last flipped dragon card

- The player will move clockwise given that the animal that they have selected matches with the animal on the tile they are standing on.

- The player will move exactly the amount of spaces (steps) that correspond to the amount of animals on the ChitCard (forward if selected same animal as the token is on and backward or counterclockwise if selected a dragon pirate)
- Given that the player selects an animal consisting of the “DragonPirate” type, they will move anticlockwise

Functional Correctness Metric 4: Changing turns to the next player

- The player’s turn is forfeited when the animal on the ChitCard they have selected does not match with the animal on the tile they are standing on
- The player that is next to them when the game first started (when all players start in their caves) out in the clockwise direction will proceed with their turn
- All 16 ChitCard’s will be flipped back to their hidden/indistinguishable state

Functional Correctness Metric 5: Winning the game

- “Winning” consists of the player returning to the cave where they started after they have proceeded through every Volcano card in the game
- The game must continue to progress with the player(s) who have returned to their cave continue forfeiting their turn until a new game has started
- The game must not continue when there is only one player left who has not returned to their cave

Functional Correctness Metric 6: Deciding on player turn order for the game duration

- Upon the initialisation of the game, all 4 DragonTokens will line up horizontally at the top of the board
- During initialisation, the first DragonToken selected will move first and will be automatically assigned to a cave
- The second DragonToken will move to the cave that is adjacent to the first DragonToken’s cave in the clockwise direction
- The third DragonToken will move to the cave that is adjacent to the second DragonToken’s cave in the clockwise direction
- The fourth DragonToken will move to the cave that is adjacent to the third DragonToken’s cave in the clockwise direction
- The player’s turns will start in the order that the DragonTokens were selected. (ie the DragonToken that was selected first will have the first turn)

Functional Appropriateness

The general definition for functional appropriateness from the international organisation for standardisation is the *degree to which the functions facilitate the accomplishment of specified tasks and objectives.*

The group’s definition for “Functional Appropriateness” is such that a program can only be considered “Functionally Appropriate” if it is of relative ease for the player to complete their required game tasks.

There is no one explicit metric that could possibly measure functional appropriateness without being overly broad and thus, redundant in its purpose. For this reason, we have

defined 5 set's of different metrics to be applied when measuring this quality characteristic. The metric to be used depends on the stated objective of the prototype.

Functional Appropriateness Metric 1: The Game should be initialised and started with minimum input and effort by any of the players. An "initialised" game is defined as each player located inside their respective caves, with the chit cards and volcanos accurately and randomly positioned around the board, with the player ready to move. The metrics for calculating this criteria include:

- The number of "clicks" needed by the player to "initialise" the game. (Within 4 clicks)
- The time (in seconds) it takes for the game to initialise. (1 second)

Functional Appropriateness Metric 2: The Player should be able to "flip" chit cards with relative ease and quickness. The metrics for calculating this criteria include:

- The number of "clicks" needed by the player to "flip over" a chit card. (1 click)
- The time (in seconds) it takes for a chit card to "flip" once clicked. (¼ of a second)

Functional Appropriateness Metric 3: The player's token should automatically move based on the previously selected "valid" chit card. The metrics for calculating the criteria include:

- The number of "clicks" needed by the Player to move their Dragon Token after the flipping of chit cards (if any). (0 clicks)
- The time (in milliseconds) taken for the player's token to move to the destination of their click (¼ of a second)

Functional Appropriateness Metric 4: A player's turn must be automatically forfeited if the player were to select an "invalid" chit card. The metrics for calculating this include:

- The number of "clicks" needed by the Player after selecting an "Invalid" chit card in order to forfeit their turn. (0 clicks)
- The time (in milliseconds) taken for the game to "switch" turns. (0.2 seconds) (the swap should be near instantaneous)

Functional Appropriateness Metric 5: A player must be defined by the game as having "won"/ "completed" the game with relative ease if they are able to reach back to their "home cave" having been around the game board. The metrics for calculating this include:

- The number of "clicks" needed by the Player to "win" / "complete" the game after having reached back to their home cave. (0 clicks)
- The time (in milliseconds) taken for the game to register the player having "won" / "completed" the game once they reach their home cave. (0.2 seconds) (the registration of the player having "won" / "completed" the game should be near instantaneous)

Appropriateness recognisability

The general definition for appropriateness recognisability from the International Organisation for Standardisation is the "Degree to which users can recognize whether a product or system is appropriate for their needs."

Our definition of appropriateness recognisability is understandability of the solution direction from its initialization.

The metrics we will apply to measure appropriateness recognisability to our Fiery Dragons implementation are:

- On first initialisation, the electronic executable resembles a physical Fiery Dragon's 'board game' via:
    - Clearly represented "player token's" that mimic those of the physical Fiery-Dragon's game
    - Obvious chit-card pieces that are of similar shape and colour to the real Fiery Dragon's game
    - Clear selection of the order of player's turn

Modifiability - (Extendibility)

The general definition for modifiability from the International Organisation for Standardisation is the "degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality"

Our definition of modifiability is the ease of extending software/code/features without creating new bugs whilst maintaining existing design patterns.

It is important to note that the modifiability characteristic does not imply, in our interpretation, that code is being replaced/deleted and updated, it is a focus on extending functionality beyond the current scope.

The following metrics used will be:

Modifiability metric 1: Number of Polymorphic Methods (polymorphism/method overriding)

- This metric counts the number of methods that exhibit polymorphic behaviour.

Modifiability metric 2: The amount of inheritance across entire source code (inheritance)

- This metric counts the number of times we inherit from classes

Modifiability metric 3: The number of abstraction across entire source code (no. abstractions)

- The metric counts the number of abstract classes we have inside the source code

Modifiability metric 4: The number of coupling across entire source code (no. coupling)

- This metric counts the number of dependencies, associations, aggregations, and compositions we have inside the source code

Using these metrics we can conclude a score given for our criteria modifiability/extensibility using the following formula:

Modifiability/Extensibility = 0.5 * no. abstractions - 0.5 * no. coupling + 0.5 * inheritance + 0.5 * polymorphism

Maintainability

The general definition for maintainability from the international organisation for standardisation is *the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements*

The group's definition of maintainability is such that the produced game model is easily adjusted to meet the changing needs of users, the environment and requirements presented.

There is no explicit metric that could possibly measure maintainability without being overly broad and thus, redundant in its purpose. For this reason, we have defined 4 set's of different metrics to be applied when measuring this quality characteristic. The metric to be used depends on the stated objective of the prototype.

Maintainability Metric 1 - DCC (direct class coupling)

- This refers to the number of classes a class is directly related to (the number of dependencies, associations, aggregations, and compositions we have inside the source code). Tighter coupling would result in changes to one area rippling through to other parts of the code making it harder to analyse the intended change of a code base. This metric also helps assess the modularity of the game such that a change to one component should have minimal impacts on other aspects of the game.

Maintainability Metric 2 - LOC (Lines of Code)

- This refers to the total number of lines in the provided codebase. A significant increase in LOC over time can make it harder to analyse and manage the codebase for potential impact on other components whilst also reducing the reusability of the code base as a large LOC value could mean that the code is extremely focused on one aspect of the game.

Using these metrics we can conclude a score given for our criteria maintainability using the following formula (lower score is better):

Maintainability = 0.5 * DCC + 0.5 * LOC

User Engagement

The general definition for user engagement from the international organisation for standardisation is the "degree to which a user interface presents functions and information in an inviting and motivating manner encouraging continued interaction."

The group's definition of user engagement is the aesthetics of the game board and overall executable. Further, we incorporate the responsiveness of the game as a factor that plays into keeping a user engaged. Each metric is scored out of 10 (higher score is better).

Metric 1 - Executable Colour selection

- The game has adopted colour combinations that are less jarring and contrasting. They fit within one of the predetermined aesthetic palettes here

Metric 2 - Game Board Piece Size Selection

- The solution should not require a user to strain his/her eyes to make out what a piece is or what it represents. This information should appear relatively quickly as the pieces and the token images should be of a large enough size to facilitate this

Metric 3 - Indication of game responsiveness

- A form of visual representation should be present when a user interacts with the game board at all times (specifically and most importantly, in between the period of taking an action (clicking) and the actual outcome desired).
- This could be a spinning wheel to indicate loading or anything within this realm of indication

Learnability

The general definition for user engagement from the international organisation for standardisation is the “degree to which the functions of a product or system can be learnt to be used by specified users within a specified amount of time”

Our definition of learnability is such that the time it takes for a user to interact with the product with intent accurately.

There is no universally accepted time frame for someone to learn how to use a piece of software as each software is different. The time we will use is solely based on our estimate of how long it should take. We approach these metrics assuming that the user already knows how the game Firey Dragon is played and all the associated rules.

Learnability Metric 1: The time it takes to understand what each image represents

- The time it takes to understand where a VolcanoCard, Cave, ChitCard, PlayerToken is located on the board

Learnability Metric 2: The time it takes to understand where the interactions on the board are

- The time it takes to figure out where all the buttons and clickable objects are

Learnability Metric 3: The time it takes to understand what all the associated interactions do:

- The time it takes to understand what the associated functionality is upon interacting with a part of the game (ie, what does this button do?)

Availability

The general definition for availability from the international organisation for standardisation is *the degree to which a system, product or component is operational and accessible when required for use.*

Our definition of “availability” is that the game must be fairly resource insensitive to run on a machine and must be available to play at any time.

There is no explicit metric that could possibly measure availability without being overly broad and thus, redundant in its purpose. For this reason, we have defined 4 set’s of different metrics to be applied when measuring this quality characteristic. The metric to be used depends on the stated objective of the prototype.

Availability Metric 1: Number of external devices needed to play

- The game should ideally only require a total of two external devices to become fully playable (1 mouse and 1 screen - apart from the required hardware)
- This metric is needed to ensure that the game is available to be run with a minimum number of external equipment needed.

Availability Metric 2: GPU Utilisation

- The game should be able to run in under 50% of total GPU utilisation for a system equipped with 4GB of GDDR5 GPU VRAM.
- Whilst running the game, the GPU Utilisation percentage will be used for subsequent score total score calculations
- This metric is needed to ensure that the provided game is available to be run in low powered machines.

Availability Metric 3: CPU Utilisation

- The game should be able to run in under 50% of total CPU utilisation for a CPU with 6 cores.
- Whilst running the game, the CPU Utilisation percentage will be used for subsequent score total score calculations
- This metric is needed to ensure that the provided game is available to be run in low powered machines.

Availability Metric 4: Mean Time before Failure

- This metric refers to the average time a system operates without failure.
- This is needed in order to identify the availability of the game whilst running it.

Using these metrics we can conclude a score given for our criteria maintainability using the following formula:

Availability = 0.25 * (1 / Number of external devices needed to play) + 0.25 * GPU Utilisation + 0.25 * CPU Utilisation + 0.25 * Mean Time before Failure

Installability

The general definition for installability from the international organisation for standardisation *is the degree of effectiveness and efficiency with which a product or system can be successfully installed and/or uninstalled in a specified environment.*

Our definition of installability is the efficiency and effectiveness of being able to open the Fiery Dragons executable in the windows/mac environment (depending on the OS that was chosen)

Installability Metric 1: Ease of downloading .exe file

- A single download package exists with the .exe file inside
- No additional supporting packages are required to download the directory with the .exe file beyond the single download

Installability Metric 2 : Ease of locating .exe file

- The exe file is within the first sub-directory of the downloaded folder. It is also among less than 10 other files in this root folder

Installability Metric 3: Efficiency of opening .exe file

- The .exe file can be opened in a single click (ie: Only need to press on the file name in the directory and it will load

Evaluation of Shannon, Ziheng, Harshath's prototypes

Shannon's Prototype

The following is an evaluation of Shannon's Prototype based on the characteristics and metrics defined above. Please note, to be able to evaluate the prototype we involved the entire technology stack, from the executable itself, to the code base and finally, the UML diagrams.

Notes for scoring: We will deduct the proportion of metric not covered by the tech-stack in the points calculation for any given quality characteristic.

Functional Completeness -

Metric 1 - "setting up the game board" was fulfilled by Shannon's game. When she initialised the executable, all 4 cave's were present, the chit-cards were laid out and the 24 animals were featured in a diamond shape.(ADHERED)

Shannon fulfilled her stated Sprint 2 functional objective of *"moving the player token based on the last selected chit-card and the current animal the token stands on".* This was functionally complete as the token moved forward or back when the associated conditions (type of animal under the chit-card) were met. (ADHERED)

The metrics measuring whether or not the player could "win the game" (metric 5) and "changing player turn" (metric 4) were not functionally complete based on her executable demonstration as the player could not return back to their home cave when the correct steps were chosen, and the next player never got the opportunity to play. (ADHERED)

Functional Completeness Metric 4, 5 and 6 - Is adequately displayed in her Sequence Diagram, whereby a player turn is stored as an attribute of the game instance and called by the state manager. (ADHERED)

For these reasons, Shannon's score for functional completeness is 6/6

Functional Correctness:

Shannon's provided functional diagram does fulfil the correctness of the primary functional objective of setting up a **random** game board. Through multiple iterations of her game board, multiple boards did not have the same chit-card configuration or volcano card configuration. It was also evident that the chit cards in this game instance were positioned in a 4x4 grid in a uniform fashion, flipping in-place and it was confirmed that one Chitcard a single flipped the same animal every time. However, there was a slight inaccuracy in such that the "flipped" dragon card hides after subsequent "turns" of the same player, where it should have been left "opened" for the entirety of the player's turn.

Shannon's game instance correctly enforced the movement of a dragon token based on the previously flipped chit-card. The movement, and the amount of movement were correctly enforced for both "progressive animals", which cause you to move "forward" in the game, as well as "Regressive" animals who move you "backwards" in the game (like Dragon pirate). However, there was a slight inaccuracy, in that the picking of a dragon token from inside the cave causes the plate rto move backwards, which should not be the case.

Because her main objective for sprint 2 was the movement of dragon tokens based on the chit card selected, she is unable to meet the following metrics in terms of software implementation: *Flipping ChitCards, Changing player turns, winning the game and deciding on the player's turn* as such a N/A will be given since we cannot assess this.

However, based on the sequence diagrams and UML diagrams we have concluded that they all fulfil the metrics that we have defined ourselves.

Based on our analysis, Shannon's functional correctness score is 5.8/6. 0.1 was deducted for the incorrect hiding of chit cards explained above, and another 0.1 was deducted for the pirate dragon causing the player to move out of their cave.

Functional Appropriateness: (Score - =3/5)

Regarding Functional Appropriateness Metric 1, Shannon's game is initialised with 1 click. It is initialised in 1 second. Shannon's prototype meets this criteria. (ADHERED)
Metric 2 - 'flipping chit card' is also adhered to. The player only needs to make 1 click to turn the card. This occurs within less than 1/10th a second. Metric 2 is fulfilled (ADHERED)
Metric 3 - It does not require a click to move Shannon's token forward/backward when selecting the correct/pirate-dragon chit-card. The movement of the player along the volcano-card's is almost instantaneous with no visual lapse in time. (ADHERED)
Metric 4 - This metric is not capable of being assessed as Shannon's game does not exhibit the functionality of changing a player's turn (N/A)
Metric 5 - This metric is not capable of being assessed as Shannon's game does not exhibit the feature of a player returning to their cave to 'win' the game. (N/A)

Appropriateness recognisability:

The only metric for measuring appropriateness was partially adhered to by Shannon's Prototype executable. There are 4 coloured distinct tokens that are of a different 'style' to that of the animals in the game. Chit cards are obvious, being uniform and in rows.

However, the chit-cards are not of a similar colour to the original Fiery Dragons game, and secondly, there is not a clear order of turns.

Thus, the overall score for this appropriateness recognisability quality characteristic is 0.5/1

Modifiability - (Extendibility)

Modifiability Metric 1 - Number of Polymorphic Methods (polymorphism/method override): 34 polymorphic methods exist within Shannon's provided codebase that get overridden *(see Abstract A for full list of polymorphic methods in Shannon's prototype)*

Modifiability Metric 2 - The amount of inheritance across entire source code (inheritance): 9 inheritance classes were found in Shannon's given codebase.

Modifiability Metric 3 - The number of abstractions across entire source code (no. abstractions): There exist 13 instances of abstractions within Shannon's code base including abstract classes, abstract methods, and interfaces.

Modifiability Metric 4 - The number of coupling across entire source code (no. coupling - refer to our definition of coupling): There exists 31 instances of coupling found in Shannon's provided documentations. This includes dependencies, associations, aggregations and compositions.

Using these metrics we can conclude a score given for Shannon's modifiability/extensibility criteria using the following formula:

Modifiability/Extensibility = 0.5 * no. abstractions - 0.5 * no. coupling + 0.5 * inheritance + 0.5 * polymorphism

= 0.5 * 13 - 0.5 * 31 + 0.5 * 9 + 0.5 * 34 = 12.5

Maintainability

Please note that for our purposes, a lower maintainability metric is valued

Maintainability Metric 1 - DCC (direct class coupling): There exists 31 instances of coupling found in Shannon's provided documentations. This includes dependencies, associations, aggregations and compositions.

Maintainability Metric 2 - LOC (Lines of Code): There exist 631 lines of code in Shannon's code base.

Shannon's Maintainability Score =0.5 * 631 lines of code + 0.5 * 31 coupling instances = 331

User Engagement

Metric 1 - Executable Colour selection: Shannon's Prototype has not got strong colour aesthetics. There are jarring colours and non-compliance with the palette guide. (3/10)

Metric 2 - Game Board Piece Size Selection : Shannon's piece's are of adequate size as a user doesn't have to strain their eyes to see what each animal is where. The point is lost because the player token is a little too close to the animal in their starting cave (difficult to distinguish) (9/10)

Metric 3 - Indication of game responsiveness: Shannon's game board is extremely responsive to user input. Flickering of dragon tokens, and absence of a loading screen decreased her score to a 7. (7/10)

Learnability

Learnability Metric 1: The time it takes to understand what each image represents

- Based on a survey of a few people in the library who were told of the way Fiery Dragon's worked, the average time to understand what each image indicates was 15 seconds.

Learnability Metric 2: The time it takes to understand where the interactions on the board are

- Based on the same survey of people in the Hargrave library who were told of the way Fiery Dragon's worked, the average time to understand where the interactions of the game were was 18 seconds.

Learnability Metric 3: The time it takes to understand what all the associated interactions do:

- In the same survey inside the Hargrave library, it took them an average of 6 seconds to understand what the interactions were.

Installability

Installability Metric 1: Ease of downloading .exe file:

- 10/10: The .exe doesn't need additional supporting packages to allow it to be downloaded

Installability Metric 2 : Ease of locating .exe file from git

- 5/10 The exe file is within the first sub-directory of the downloaded folder. However, it is among more than 10 other files in this root folder

Installability Metric 3: Efficiency of opening .exe file

- 10/10: The .exe file can be opened in a single click (ie: Only need to press on the file name in the directory and it will load

Availability

Availability Metric 1: 2

- The game only requires a total of two external devices to become fully playable (1 mouse and 1 screen - apart from the required hardware)

Availability Metric 2: GPU Utilisation: 4%

- Whilst running the game, the GPU Utilisation percentage was 4%

Availability Metric 3: CPU Utilisation - 0.7%

- The CPU utilisation of Shannon's executable is 0.7% on average. The screenshot of this is provided in appendix B.

- Availability Metric 4: N/A - The game did not crash in our tests. (a score of 1 is used for calculations for this regard)

Using these metrics we can conclude a score given for our criteria availability using the following formula:

Availability = 0.25 * (1 / Number of external devices needed to play) + 0.25 * GPU Utilisation + 0.25 * CPU Utilisation + 0.25 * Mean Time before Failure

0.25 * 0.5 + 0.25 * 0.04 + 0.25 * 0.007 + 0.25 * 1 = 0.39

Ziheng's Prototype

The following is an evaluation of Ziheng's Prototype based on the characteristics and metrics defined above. Please note, to be able to evaluate the prototype we involved the entire technology stack, from the executable itself, to the code base and finally, the UML diagrams.

Notes for scoring: We will deduct the proportion of metric not covered by the tech-stack in the points calculation for any given quality characteristic.

Functional Completeness -

Metric 1 - "setting up the game board" was fulfilled by Ziheng's game. When he initialised the executable, all 4 cave's were present, the chit-cards were laid out and the 24 animals were featured in a square shape.(ADHERED)

Ziheng did not fulfil his stated Sprint 2 functional objective of *"moving the player token based on the last selected chit-card and the current animal the token stands on"* in the executable however it is demonstrated in the sequence diagram. (ADHERED)

The metrics measuring whether or not the player could "win the game" (metric 5) was NOT functionally complete based on Ziheng's executable demonstration as the player could not return back to their home cave when the correct steps were chosen, however it is demonstrated in the sequence diagram. (ADHERED)

Metric 4 was fulfilled as the player can change when selecting an incorrect chit-card. (ADHERED)

Functional Completeness Metric 1-5 is adequately displayed in Ziheng's Sequence Diagram, whereby a player turn is stored as an attribute of the game instance and called by the state manager. (ADHERED)

Based on our analysis, Ziheng's functional correctness score is 5.0/6 as there is no notion of 'choosing the order of players' at the start of the game (last metric)

For these reasons, Ziheng's score for functional completeness is 5/6

Functional Correctness:

Ziheng's provided functional diagram fulfils the correctness of the primary functional objective of setting up a **random** game board. Through multiple iterations of Ziheng's game board, multiple boards did not have the same chit-card configuration or volcano card

configuration. However there was no notion of a chitcard flipping in place. No notion of a chit card staying open for the duration of a player's turn.

Because his main objective for sprint 2 was the *change of player turn*, Ziheng is unable to meet the following metrics in terms of software implementation: *Flipping ChitCards, moving the player, winning the game* as such a N/A will be given since we cannot assess this.

However, based on the sequence diagrams and UML diagrams we have concluded that they all fulfil the metrics that we have defined ourselves.

Based on our analysis, Ziheng's functional correctness score is 5.0/6 as there is no notion of 'choosing the order of players' at the start of the game (last metric)

Functional Appropriateness: (2/5)

Regarding Functional Appropriateness Metric 1, Ziheng's game is initialised with 1 click. It is initialised in 1 second. Ziheng's prototype meets this criteria. (ADHERED)
Metric 2 - 'flipping chit card' is also adhered to. The player only needs to make 1 click to turn the card. This occurs within less than 1/10th a second. Metric 2 is fulfilled (N/A)
Metric 3 - It does not require a click to move Ziheng's token forward/backward when selecting the correct/pirate-dragon chit-card. The movement of the player along the volcano-card's is almost instantaneous with no visual lapse in time. (N/A)
Metric 4 - The turn is automatically forfeited and transferred to the subsequent player when the player choosing an "invalid" chit-card in Ziheng's game instance (ADHERED)
Metric 5 - This metric is not capable of being assessed as Ziheng's game does not exhibit the feature of a player returning to their cave to 'win' the game. (N/A)

Appropriateness recognisability:

The only metric for measuring appropriateness was partially adhered to by Ziheng's Prototype executable. Chit Cards are easily recognisable in the middle of the screen, however they weren't in the shapes of cards as such they were not immediately obvious at first glance. Although the player tokens were identifiable the board unfortunately didn't mimic the board accurately as the board seemed almost as if disconnected to the cave.

Thus, the overall score for this appropriateness recognisability quality characteristic is 0.75/1

Modifiability - (Extendibility)

Modifiability Metric 1 - Number of Polymorphic Methods (polymorphism/method override): 30 polymorphic methods exist within Ziheng's provided codebase that get overridden *(see Abstract C for full list of polymorphic methods in Ziheng's prototype)*

Modifiability Metric 2 - The amount of inheritance across the entire source code (inheritance): 7 inheritance classes were found in Ziheng's given codebase.

Modifiability Metric 3 - The number of abstractions across entire source code (no. abstractions): There exist 2 instances of abstractions within Ziheng'scode base including abstract classes, abstract methods, and interfaces.

Modifiability Metric 4 - The number of coupling across entire source code (no. coupling - refer to our definition of coupling): There exists 20 instances of coupling found in Ziheng's

provided documentations. This includes dependencies, associations, aggregations and compositions.

Using these metrics we can conclude a score given for Ziheng's modifiability/extensibility criteria using the following formula:

Modifiability/Extensibility = 0.5 * no. abstractions - 0.5 * no. coupling + 0.5 * inheritance + 0.5 * polymorphism

= 0.5 * 2 - 0.5 * 20 + 0.5 * 7 + 0.5 * 30 = 9.5

Maintainability

Please note that for our purposes, a lower maintainability metric is valued

Maintainability Metric 1 - DCC (direct class coupling): There exists 20 instances of coupling found in Ziheng's provided documentations. This includes dependencies, associations, aggregations and compositions.

Maintainability Metric 2 - LOC (Lines of Code): There exist 596 lines of code in Ziheng's code base.

Ziheng's Maintainability Score =0.5 * 596 lines of code + 0.5 * 20 coupling instances = 308

User Engagement

Metric 1 - Executable Colour selection: Ziheng's colour selection was unimaginative, using only 2 tones. For the majority of the game. (5/10)

Metric 2 - Game Board Piece Size Selection : Harshath's board size was appropriate for the screen size. (9/10)

Metric 3 - Indication of game responsiveness: Harshath's game board is extremely responsive to user input. (10/10)

Learnability

Learnability Metric 1: The time it takes to understand what each image represents

- During the initial opening of the game it took Harshath and Shannon an average of 20 seconds to understand what each image represented

Learnability Metric 2: The time it takes to understand where the interactions on the board are

- Harshath and Shannon took an average of 30 seconds to understand where each interaction (button) was (it took a little longer given that the chit-cards themselves were on the right hand side, not in the middle of the screen like standard).

Learnability Metric 3: The time it takes to understand what all the associated interactions do:

- Harshath and Shannon took 5 seconds to understand what all the interactions did.

Installability

Installability Metric 1: Ease of downloading .exe file:

- 10/10: The .exe doesn't need additional supporting packages to allow it to be downloaded

Installability Metric 2 : Ease of locating .exe file from git

- 10/10. Ziheng's executable file was easy to locate within his git repository as it was found on his main folder.

Installability Metric 3: Efficiency of opening .exe file

- 10/10: The .exe file can be opened in a single click (ie: Only need to press on the file name in the directory and it will load

Availability

Availability Metric 1: 2

- The game only requires a total of two external devices to become fully playable (1 mouse and 1 screen - apart from the required hardware)

Availability Metric 2: GPU Utilisation: 0%

- Whilst running the game, the GPU Utilisation percentage was 0%

Availability Metric 3: CPU Utilisation - 3.6%

- The CPU utilisation of Ziheng's executable is 3.5% on average. The screenshot of this is provided in appendix B.
- Availability Metric 4: N/A - The game did not crash in our tests. (a score of 1 is used for calculations for this regard)

Using these metrics we can conclude a score given for our criteria availability using the following formula:

Availability = 0.25 * (1 / Number of external devices needed to play) + 0.25 * GPU Utilisation + 0.25 * CPU Utilisation + 0.25 * Mean Time before Failure

0.25 * 0.5 + 0.25 * 0 + 0.25 * 0.036+ 0.25 * 1= 0.384

Harshath's Prototype

The following is an evaluation of Harshath's Prototype based on the characteristics and metrics defined above. Please note, to be able to evaluate the prototype we involved the entire technology stack, from the executable itself, to the code base and finally, the UML diagrams.

Notes for scoring: We will deduct the proportion of metric not covered by the tech-stack in the points calculation for any given quality characteristic.

Functional Completeness -

Metric 1 - "setting up the game board" was fulfilled by Harsharth's game. When heinitialised the executable, all 4 cave's were present, the chit-cards were laid out and the 24 animals were featured in a diamond shape.(ADHERED)

Harsh fulfilled his stated Sprint 2 functional objective of *"moving the player token based on the last selected chit-card and the current animal the token stands on".* This was functionally complete as the token moved forward or back when the associated conditions (type of animal under the chit-card) were met. (ADHERED)

The metrics measuring whether or not the player could "win the game" (metric 5) and "changing player turn" (metric 4) were not functionally complete based on Harsh's executable demonstration as the player could not return back to their home cave when the correct steps were chosen, and the next player never got the opportunity to play. (ADHERED)

Functional Completeness Metric 4, 5 and 6 - Is adequately displayed in Harsh's Sequence Diagram, whereby a player turn is stored as an attribute of the game instance and called by the state manager. (ADHERED)

For these reasons, Harshath's score for functional completeness is 6/6

Functional Correctness:

Harthath's provided functional diagram fulfils the correctness of the primary functional objective of setting up a **random** game board. Through multiple iterations of Harh's game board, multiple boards did not have the same chit-card configuration or volcano card configuration. However there was no notion of a chitcard flipping in place. No notion of a chit card staying open for the duration of a player's turn.

Harsh's game instance correctly enforced the movement of a dragon token based on the previously flipped chit-card. The movement, and the amount of movement were correctly enforced for both "progressive animals", which cause you to move "forward" in the game, as well as "Regressive" animals who move you "backwards" in the game (like Dragon pirate).

Because his main objective for sprint 2 was the *movement of dragon tokens based on the chit card selected*, Harsh is unable to meet the following metrics in terms of software implementation: *Flipping ChitCards, Changing player turns, winning the game and deciding on the player's turn* as such a N/A will be given since we cannot assess this.

However, based on the sequence diagrams and UML diagrams we have concluded that they all fulfil the metrics that we have defined ourselves.

Based on our analysis, Harshath functional correctness score is 5.0/6. 0.5 was deducted for the incorrect hiding of chit cards explained above, and 0.5 deducted for not showing how chit cards flip in place when clicked.

Functional Appropriateness: (3/5)

Regarding Functional Appropriateness Metric 1, Harshath's game is initialised with 1 click. It is initialised in 1 second. Harshath's prototype meets this criteria. (ADHERED)
Metric 2 - 'flipping chit card' is also adhered to. The player only needs to make 1 click to turn the card. This occurs within less than 1/10th a second. Metric 2 is fulfilled (ADHERED)
Metric 3 - It does not require a click to move Harshath's token forward/backward when selecting the correct/pirate-dragon chit-card. The movement of the player along the volcano-card's is almost instantaneous with no visual lapse in time. (ADHERED)
Metric 4 - This metric is not capable of being assessed as Harshath's game does not exhibit the functionality of changing a player's turn (N/A)
Metric 5 - This metric is not capable of being assessed as Harshath's game does not exhibit the feature of a player returning to their cave to 'win' the game. (N/A)

Appropriateness recognisability:

The only metric for measuring appropriateness was partially adhered to by Harhath's Prototype executable. There are 4 coloured distinct tokens that are of a different 'style' to that of the animals in the game. Chit cards are obvious, being uniform and in rows.
Chit-cards are also of a similar colour to the original Fiery Dragons game,
However, there is not a clear order of turns.

Thus, the overall score for this appropriateness recognisability quality characteristic is 0.75/1

Modifiability - (Extendibility)

Modifiability Metric 1 - Number of Polymorphic Methods (polymorphism/method override): 37 polymorphic methods exist within Harsh's provided codebase that get overridden *(see Abstract C for full list of polymorphic methods in Harsh's prototype)*

Modifiability Metric 2 - The amount of inheritance across the entire source code (inheritance): 20 inheritance classes were found in Harsh's given codebase.

Modifiability Metric 3 - The number of abstractions across entire source code (no. abstractions): There exist 16 instances of abstractions within Harh's code base including abstract classes, abstract methods, and interfaces.

Modifiability Metric 4 - The number of coupling across entire source code (no. coupling - refer to our definition of coupling): There exists 25 instances of coupling found in Harshath's provided documentations. This includes dependencies, associations, aggregations and compositions.

Using these metrics we can conclude a score given for Harshath's modifiability/extensibility criteria using the following formula:

Modifiability/Extensibility = 0.5 * no. abstractions - 0.5 * no. coupling + 0.5 * inheritance + 0.5 * polymorphism

= 0.5 * 16 - 0.5 * 25 + 0.5 * 20 + 0.5 * 37 = 24

Maintainability

Please note that for our purposes, a lower maintainability metric is valued

Maintainability Metric 1 - DCC (direct class coupling): There exists 25 instances of coupling found in Harsh's provided documentations. This includes dependencies, associations, aggregations and compositions.

Maintainability Metric 2 - LOC (Lines of Code): There exist 1275 lines of code in Harsh's code base.

Harsh's Maintainability Score =0.5 * 1275 lines of code + 0.5 * 25 coupling instances = 650

User Engagement

Metric 1 - Executable Colour selection: Harshath's colour selection was specifically chosen so that it was aesthetically pleasing. The soft contrast between colours made it very easy to distinguish what was what whilst maintaining a pleasing visual of the board. (10/10)

Metric 2 - Game Board Piece Size Selection : Harshath's board size was appropriate for the screen size. The only drawback is the animals inside the tiles where it is slightly small. (8/10)

Metric 3 - Indication of game responsiveness: Harshath's game board is extremely responsive to user input. (10/10)

Learnability

Learnability Metric 1: The time it takes to understand what each image represents

- During the initial opening of the game it took Ziheng and Shannon an average of 20 seconds to understand what each image represented

Learnability Metric 2: The time it takes to understand where the interactions on the board are

- Ziheng and Shannon took an average of 30 seconds to understand where each interaction (button) was (it took a little longer given that the chit-cards themselves were on the right hand side, not in the middle of the screen like standard).

Learnability Metric 3: The time it takes to understand what all the associated interactions do:

- Ziheng and Shannon took 6 seconds to understand what all the interactions (button)

Installability

Installability Metric 1: Ease of downloading .exe file:

- 10/10: The .exe doesn't need additional supporting packages to allow it to be downloaded

Installability Metric 2 : Ease of locating .exe file from git

- 8/10 The exe file is within the dist folder, however because it isn't a file but a folder since it is a .app executable, it wasn't immediately obvious that this was the exe file.

Installability Metric 3: Efficiency of opening .exe file

- 10/10: The .exe file can be opened in a single click (ie: Only need to press on the file name in the directory and it will load

Availability

Availability Metric 1: 2

- The game only requires a total of two external devices to become fully playable (1 mouse and 1 screen - apart from the required hardware)

Availability Metric 2: GPU Utilisation: 0%

- Whilst running the game, the GPU Utilisation percentage was 0%

Availability Metric 3: CPU Utilisation - 4.5%

- The CPU utilisation of Harsh's executable is 4.5% on average. The screenshot of this is provided in appendix B.
- Availability Metric 4: N/A - The game did not crash in our tests. (a score of 1 is used for calculations for this regard)

Using these metrics we can conclude a score given for our criteria availability using the following formula:

Availability = 0.25 * (1 / Number of external devices needed to play) + 0.25 * GPU Utilisation + 0.25 * CPU Utilisation + 0.25 * Mean Time before Failure

0.25 * 0.5 + 0.25 * 0 + 0.25 * 0.045+ 0.25 * 1= 0.38625

Shannon’s Key Findings

There were a few core components of Shannon’s prototype that are worth describing. These components are polarising, in that they were either extremely well fulfilled based on our quality assessment of them, or poorly fulfilled: below the quality threshold of Harsh/Ziheng’s same components.

Firstly, let's discuss the features that excelled in her quality assessment.
Shannon’s game board is almost fully functionally complete in its executable initialisation. The token can advance or regress around the volcano ring at the selection of a chit card. Chit-cards are capable of being ‘turned’ over and visually, identical to one another. The game features the correct methodology of a player ‘skipping’ the cave it is not allowed to enter. While the proposed requirements of having players change their turn and having players “win” the game are not explicit in the executable, they have been well thought of and displayed in the sequence diagrams.

The functional completeness and correctness will be highly beneficial when attempting to create Sprint 3, as we are assured that Shannon’s code as it stands produces accurate results that meet the requirements of the Fiery Dragons game.

Another driving success was the functional appropriateness of Shannon’s prototype. It is very quick and simple to make a player move, to select a chit-card and start the game. Each action takes one or less clicks and occurs within half a second. We do not need to work on her time complexity or the logic design to make interactions.

Shannon’s maintainability score was quite low, a good thing. This is primarily driven by her relatively now lines of code and low coupling score. Her design supports maintainability into Sprint 3, as we add more code and attempt to alter her existing base, we don’t have to trawl through so much content to make fixes

Finally, we see that Shannon’s game is quite learnable, given that it took survey participants less than 2 minutes to explain to us how they would attempt to play the game and where the buttons were on the screen.

Conversely, let us discuss some of the drawbacks to Shannon’s prototype as identified in our evaluation.

Firstly, we notice that the game does not exhibit in its executable, nor the class diagrams that the chit-card should remain ‘open’ throughout the duration of a player’s turn and not re-clickable for their turn duration. We will be sure to consider this draw-back when creating our third sprint.

The next lowest quality aspect to the game is the visual aesthetic (user engagement). Colour selection was poor (did not adhere to any defined colour palette) and the token’s flicker each time an action is taken (button is clicked). Finally, we notice that the chit-cards do not really resemble the chit-cards within the original Fiery Dragons game (to improve this, they should ideally be coloured a light brown or cream rather than red).

Finally, we will be sure to consider the lower level of modifiability (extendability) present in Shannon’s code base. She had a relatively lower score than that of Ziheng and Harshath, and thus, might make our Sprint 3 extensions more work. We will need to pick the classes and methods that are less coupled.

Ziheng’s Key Findings

Ziheng’s prototype contains mixed reviews. There are qualities that are sufficient and meet the requirements, however there are also parts of the prototype that are quite lacking.

Starting off with the sufficient parts of the requirement, Ziheng’s functional completeness meets the following metrics except for choosing the order of players during board initialisation. Adding on, Ziheng’s software prototype successfully completes his sprint 2 goal of changing players and randomising the chit cards upon initialisation, however it is lacking in the aspect of flipping chit cards, moving players around the board, winning the game and deciding which player goes first.

Moving onto functional correctness, inside Ziheng’s sprint 2 software prototype, he is able to demonstrate successfully that players' turns are forfeited given that they select the incorrect chit card and upon selecting the correct chit card, they get another turn. For other functionalities, he demonstrates such capabilities through his sequence diagram to which they suffice in correctness as they follow the metrics as defined earlier in the document. An area that did not follow the metrics and cannot be assessed against is the decision on the order of which a player proceeds with.

In terms of functional appropriateness, Ziheng received a score of 2/5 as he was unable to demonstrate the metrics that displayed functional appropriateness such as a player returning to their cave to “win the game”.

Addressing appropriateness recognisability, Ziheng’s board was once again only partially successful in fulfilling this criteria. His chit cards were obvious as they were placed in the middle of the board with the pitfall of his chit cards not representing chit cards.

Discussing modifiability/extensibility, Ziheng scored quite low using our metric suggesting that his code is not very extensible. A major contributing factor was the lack of abstractions.

Ziheng’s maintainability score was 308 calculated using our criteria metric devised by us where the lower the result is the easier it is to maintain. This is a good score suggesting that Ziheng’s code is easily maintainable.

The drawbacks to Ziheng’s software implementation is the user engagement. Upon opening, the colour features do not stand out aesthetically. Colour palette does not match and it is hard to be engaged in the game as visuals do not stand out.

In terms of learnability, the game was quite easy to follow and understand what was what due to the simplicity of the software prototype displayed.

Installability also scored quite high as there was no prior setting up or installation of any packages beforehand to get the game set up.

Finally, availability scored quite high as the game being presented did not demand top end hardware or the latest operating system to run.

Harshath’s Key Findings

Harshath’s prototype had good reviews overall. The reviews were not as polarsing compared to a few others, with components either extremely well received by peer reviewers, or marked in the “average” range compared to the other prototypes.

Looking at the “well received” aspects of the game submission first, it was found that Harshath’s prototype adhered to all functional completeness metrics, receiving full marks for that section. This shows that Harshath’s prototype had fully functional components, making it easier to build off of this game prototype for Sprint 3 if needed. Harshath meets the required appropriateness for the implemented criteria as well, adhering the the required metric (unimplemented metrics could not be tested due to the way assessment criteria is defined)

Looking modifiability, Harshath had the highest modifiability score compared to other other two prototypes, suggesting that this prototype game instance was great to build off of during Sprint 3 as the base code is highly extensible for future game installations in Sprint 4. User Engagement metrics for Harshath’s prototype was also ranked the highest compared to the other two with Harshath scoring full marks for Metric 1 and 3 suggesting that this prototype has great aesthetics and game responsiveness. An area of improvement was found with Metric 2 (scoring 8/10) suggesting that members would like to see bigger game “elements” as they found the current “game elements” hard to see.

Looking now at Learnability metrics, Harshath’s game prototype had one of the best learnability metric evaluation compared to the other prototypes, suggesting that members found it much easier to understand the game board instance (with previous knowledge of the Fiery Dragons game) compared to other prototypes. One feedback worth mentioning was that “due to the similarities between the provided game board to the actual game board” the game instance provided was much easier to understand and play.

Looking at installability metrics, Harshath’s game instance runs as required (as an exe), without any need for downloading extra dependencies and with ease of a single click. A point was made to ensure that Harshath’s game executable file should be easier to find within the git repository.

Looking at availability metrics, Harshath’s game instance again fulfils the required criteria, requiring minimum hardware capabilities to run.

Now, looking at the not so “well received" aspects of the game, the maintainability of Harshath’s provided codebase ranked low compared to other prototypes due to the large number of lines of code in the codebase. This could make it harder for other team members to understand Harshath’s codebase (for Sprint 3 implementation purposes). Whilst the lines of code in the codebase was quite high, the amount of coupling between classes remained on-par with the other code bases (Shannon’s and Ziheng’s), suggesting that Harshath’s codebase remains extensible and easy to build off of.

Consolidated solution we are using for sprint 3

1. Set up the initial game board - Harshath

We decided to use Harshath's section of code for a variety of reasons. One of the reasons was user experience. Harshath's board best represented how an actual physical board would look. In addition, the colour scheme best matched the board and was overall appealing to the eye.

Another reason was the extensibility to it if there were more volcano cards to be added or even chit cards. Because the design of the board is circular, utilising degrees as the primary method of location identification, it is easier to add more volcano cards compared to if it were in any other shape, in our case, square (Ziheng) and diamond (Shannon). Furthermore, it prevents the need to hardcode the position of such volcano cards which further follow the open-close principle.

2. Flipping of dragon ("chit") cards - Harshath/Shannon/Ziheng

Shannnon's idea of having "state" classes to handle varying events in the game such as a "clicked" event or a "quit" event was adopted in our final combined solution approach.

The approach helped to reduce the 'god' like class that GameRunning.py could have become if it was not divided into its own separate class entity.

All the logic that comes with flipping the card and moving the player on the board is handled inside the ClickedState.py class (more details on player moving later).

Expanding on the flipping card inside the "ClickedState.py", Ziheng, Shannon and Harsh all had a similar idea on how to implement flipping cards.
The flipping card implementation consists of having an attribute to tell us whether that chit card has been clicked (via the "MouseDown" event in pygame), and where this is the case, calling the flip() method on this chit-card object.

3. Movement of dragon tokens based on their current position as well as the last flipped dragon card - Harshath/Shannon

Because the movement of a player is dependent on the chitcard that was flipped, we have to combine the implementation of movement with Shannon's idea of a "state" to handle a chitcard being clicked.
We ultimately decided on Harshath's movement of players as we are using his design for setting up the game board which best already synergises the display of the player being displayed on the board and the nature of "stepping forward"/around the circle by a number of radian degrees

Shannon and Ziheng's idea of how to display the player being moved to a new tile won't work because the shapes of their board (diamond and square respectively) doesn't work since the positioning of the tiles and coordinate system isn't compatible with Harshath's circular board.

To focus more specifically, we are implementing the double dispatching pattern of all 3 (Shannon, Ziheng and Harshath's code base). This approach allowed the selected chit-card to be compared to the current animal a player token stands on in a fashion that doesn't rely on extensive if, else statements.
Harsh and Shannon both used a similar (slightly different implementation style) logic to deal with the concept of "skipping a cave" that isn't the player's "home". We opted for Harshath's approach of calling gameProgress.forward() twice due to the cleaner logic for extensibility and code maintainability.

4. Change of turn to the next player - Shannon

We ultimately decided to use Shannon's idea of a circular list to implement the idea of the next player. We rejected Ziheng's idea of having such logic inside the main game loop as that broke single responsibility. Despite that it was simple in terms of how many lines of code needed to implement such functionality, it contributed to the problem of creating a god class.

In saying that we will use Shannon's idea of handling this in a separate class outside of the main game loop which is solely dedicated to switching the turn of players. Harshath's current design of having the next Player handled by the player itself breaks the responsibility principle as the next player should be handled by a PlayerManager class. Thus with a PlayerManager we only need to initialise it once and can handle any next player turn via the next_player() method.

Ziheng's clever use of the window screen header to display the current player's turn was adopted in the final solution design. We agreed on adopting this feature in an effort to make our game more usable and boost the metrics within this quality characteristic.

5. Winning the game

We hadn't had a member of our team implement this feature in the previous Sprint. For this reason, we have had to adopt our own unique approach.

Rather than creating a 'winning manager' class. We opted to implement a conditional approach within the logic used to move a player Forward() or backward().
We agreed that the most maintainable approach would be clean and understandable code. We adopted a 2 conditional approach if a player is "atHome" (a function that checks the player's home Cave attribute and whether a player is located in a cave via the check_cave() method.

I can justify the choice of integrating winning within the player movement logic because of its benefits to code readability for someone who might intend to extend the code we made today in the future. I recognise that the approach might violate the single responsibility principle, whereby the ClickedState() class is now covering more aspects of the game requirements.

UML Class Diagram

CRC Cards

A CRC card is a tool to facilitate the design of a software solution. It provides the class name, responsibilities and collaborators for each of the classes in design (based on the UML design class diagram provided above).

ChitCard	
Knows its animal - *getAnimal()*	Animal
Assigning animal to itself - *setAnimal()*	Animal
Retrieving the respective png image - *getClosedImage()*	
Know whether it has been flipped - *isFlipped()*	
Know how many animals on itself - *getNumber()*	
To flip itself when necessary - *flip()*	

Purpose:
The purpose of the ChitCard is to allow the player to interact with it to progress through the game. As such, we will need to be able to click on it to flip it and based on the animal on that ChitCard, will determine whether we progress or not.

PlayerManager	
Be able to add new players to manage - *add_player()*	Player
Retrieve the next player - *next_player()*	Player
Retrieve the current player - *get_current_player()*	Player
Get all players other than the current player - *get_other_players()*	Player
Remove the current player - *remove_current_player()*	Player

Purpose:
The purpose of the PlayerManager is so that we can manage who's turn in the game it is. With this we can be flexible even if there aren't 4 players trying to play the game as we can

add and remove Players at will. We can also identify the next Player's turn with a manager and give us a layer of abstraction.

Player	
Retrieve the name of the player - *getName()*	StartGame
Get the image representing the player - *getImage()*	StartGame
Get where the starting point of the player was - *gethomeTileAngle()*	
Know where it is on the board - *getProgression()*	GameProgression
Set name of player - *setName()*	
Set the png file to the player - *setImage()*	
Set the initial angle of where it starts off in the board - *sethomeAngle()*	
Assign the progression of the player - *setProgression*	GameProgression
Determine whether it has arrived home or not - *isHome()*	
Determine if the tile is already occupied - *isOccupied()*	Player

Purpose:
The purpose of the Player is to represent the actual player playing. As such, the Player class should be represented by a Dragon Token image, the player should know where it started on the board, meaning it knows where the cave it started in was. Therefore, it should know when it has reached back home.

Board	
Be able to draw the screen - *draw()*	ChitCard, Volcano, Player, Cave
Display the winning player - *drawWinning()*	Player

Purpose:
The purpose of the board is to provide a board for the game to be run. This includes storing the actual attributes such as volcanos inside the board as well as visually displaying them onto the screen.

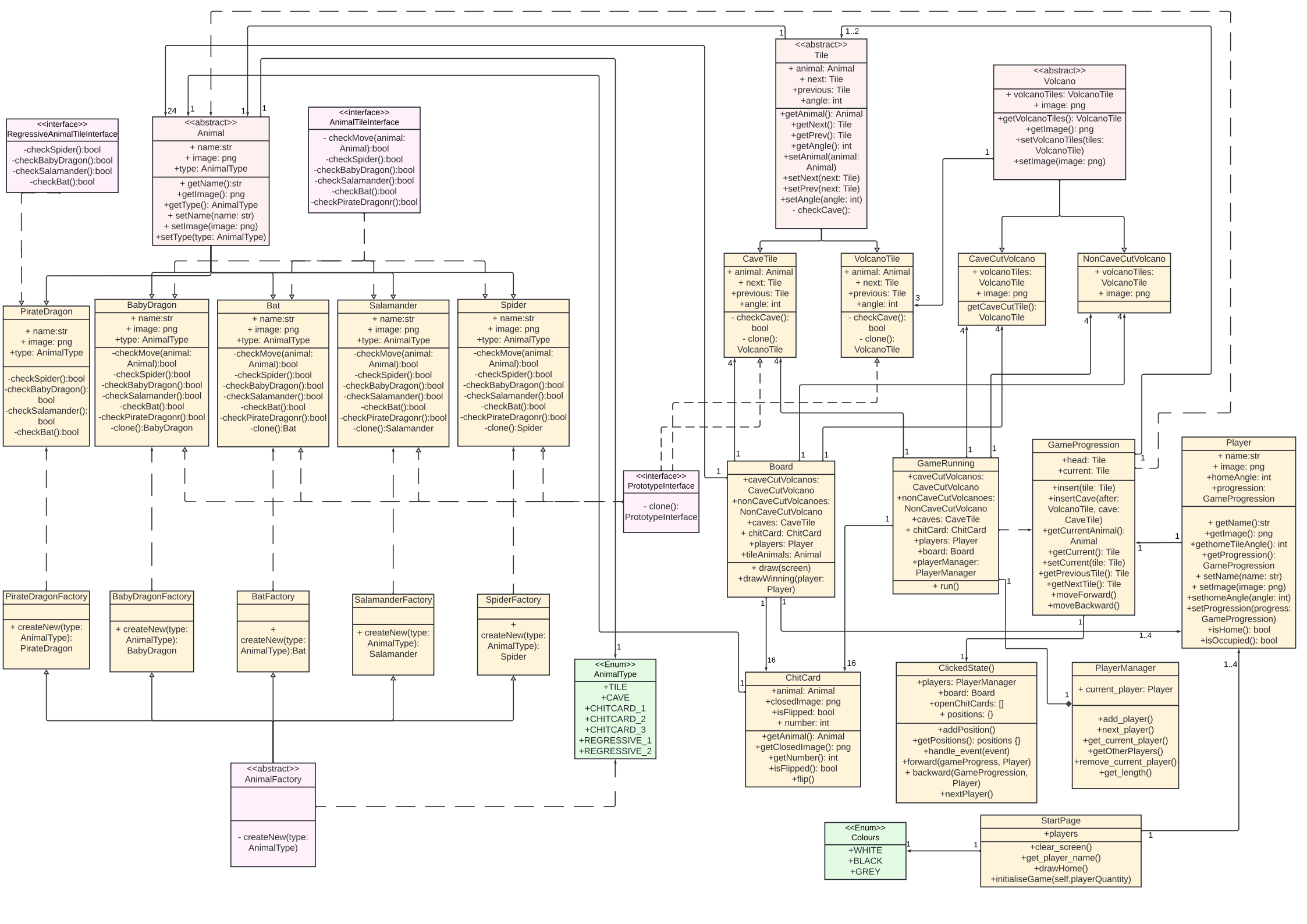
GameProgression

Insert tiles - *insert()*	Tile
Insert caves - *insertCave()*	CaveTile
Get the animal on the current tile the player is on - *getCurrentAnimal()*	Animal, Tile
Set the current tile the player is on - *setCurrent()*	Tile
Get the previous tile to the current tile the player is on - *getPreviousTile()*	Tile
Get the next tile to the current tile the player is on - *getNextTile()*	Tile
Set the next tile as the current tile - *moveForward()*	Tile
Set the previous tile as the current tile - *moveBackward()*	Tile

Purpose:
The purpose of the GameProgression is to handle the logic of moving the players. This includes moving the player, determining the animal that the player is standing on to compare with the chit card selected. All logic that is needed to progress with the game is handled by GameProgression.

StartPage	
Set the screen to a blank canvas - *clear_screen()*	
Get the player who will be playing's name - *get_player_name()*	
Display the home screen of the start page - *drawHome()*	
Initialise the game - *initialiseGame()*	GameRunning

Purpose:
The purpose of StartPage is to display a start page for the players to insert their names, choose the amount of players playing and the order of their turn as well. This also includes displaying the actual intractable window onto the display as well.


<<interface>>
RegressiveAnimalTileInterface
-checkSpider():bool
-checkBabyDragon():bool
-checkSalamander():bool
-checkBat():bool
<<abstract>>
Animal
+ name:str
+ image: png
+type: AnimalType
+ getName():str
+getImage(): png
+getType(): AnimalType
+ setName(name: str)
+ setImage(image: png)
+setType(type: AnimalType)
<<interface>>
AnimalTileInterface
- checkMove(animal: Animal):bool
-checkSpider():bool
-checkBabyDragon():bool
-checkSalamander():bool
-checkBat():bool
-checkPirateDragonr():bool
<<abstract>>
Tile
+ animal: Animal
+ next: Tile
+previous: Tile
+angle: int
+getAnimal(): Animal
+getNext(): Tile
+getPrev(): Tile
+getAngle(): int
+setAnimal(animal: Animal)
+setNext(next: Tile)
+setPrev(next: Tile)
+setAngle(angle: int)
- checkCave():
<<abstract>>
Volcano
+ volcanoTiles: VolcanoTile
+ image: png
+getVolcanoTiles(): VolcanoTile
+getImage(): png
+setVolcanoTiles(tiles: VolcanoTile)
+setImage(image: png)
CaveTile
+ animal: Animal
+ next: Tile
+previous: Tile
+angle: int
- checkCave(): bool
- clone(): VolcanoTile
VolcanoTile
+ animal: Animal
+ next: Tile
+previous: Tile
+angle: int
- checkCave(): bool
- clone(): VolcanoTile
CaveCutVolcano
+ volcanoTiles: VolcanoTile
+ image: png
getCaveCutTile(): VolcanoTile
NonCaveCutVolcano
+ volcanoTiles: VolcanoTile
+ image: png
PirateDragon
+ name:str
+ image: png
+type: AnimalType
-checkSpider():bool
-checkBabyDragon():bool
-checkSalamander():bool
-checkBat():bool
BabyDragon
+ name:str
+ image: png
+type: AnimalType
-checkMove(animal: Animal):bool
-checkSpider():bool
-checkBabyDragon():bool
-checkSalamander():bool
-checkBat():bool
-checkPirateDragonr():bool
-clone():BabyDragon
Bat
+ name:str
+ image: png
+type: AnimalType
-checkMove(animal: Animal):bool
-checkSpider():bool
-checkBabyDragon():bool
-checkSalamander():bool
-checkBat():bool
-checkPirateDragonr():bool
-clone():Bat
Salamander
+ name:str
+ image: png
+type: AnimalType
-checkMove(animal: Animal):bool
-checkSpider():bool
-checkBabyDragon():bool
-checkSalamander():bool
-checkBat():bool
-checkPirateDragonr():bool
-clone():Salamander
Spider
+ name:str
+ image: png
+type: AnimalType
-checkMove(animal: Animal):bool
-checkSpider():bool
-checkBabyDragon():bool
-checkSalamander():bool
-checkBat():bool
-checkPirateDragonr():bool
-clone():Spider
<<interface>>
PrototypeInterface
- clone(): PrototypeInterface
Board
+caveCutVolcanos: CaveCutVolcano
+nonCaveCutVolcanoes: NonCaveCutVolcano
+caves: CaveTile
+ chitCard: ChitCard
+players: Player
+tileAnimals: Animal
+ draw(screen)
+drawWinning(player: Player)
GameRunning
+caveCutVolcanos: CaveCutVolcano
+nonCaveCutVolcanoes: NonCaveCutVolcano
+caves: CaveTile
+ chitCard: ChitCard
+players: Player
+board: Board
+playerManager: PlayerManager
+ run()
GameProgression
+head: Tile
+current: Tile
+insert(tile: Tile)
+insertCave(after: VolcanoTile, cave: CaveTile)
+getCurrentAnimal(): Animal
+getCurrent(): Tile
+setCurrent(tile: Tile)
+getPreviousTile(): Tile
+getNextTile(): Tile
+moveForward()
+moveBackward()
Player
+ name:str
+ image: png
+homeAngle: int
+progression: GameProgression
+ getName():str
+getImage(): png
+gethomeTileAngle(): int
+getProgression(): GameProgression
+ setName(name: str)
+ setImage(image: png)
+sethomeAngle(angle: int)
+setProgression(progress: GameProgression)
+isHome(): bool
+isOccupied(): bool
PirateDragonFactory
+ createNew(type: AnimalType): PirateDragon
BabyDragonFactory
+ createNew(type: AnimalType): BabyDragon
BatFactory
+ createNew(type: AnimalType):Bat
SalamanderFactory
+ createNew(type: AnimalType): Salamander
SpiderFactory
+ createNew(type: AnimalType): Spider
<<Enum>>
AnimalType
+TILE
+CAVE
+CHITCARD_1
+CHITCARD_2
+CHITCARD_3
+REGRESSIVE_1
+REGRESSIVE_2
ChitCard
+animal: Animal
+closedImage: png
+isFlipped: bool
+ number: int
+getAnimal(): Animal
+getClosedImage(): png
+getNumber(): int
+isFlipped(): bool
+flip()
ClickedState()
+players: PlayerManager
+board: Board
+openChitCards: []
+ positions: {}
+addPosition()
+getPositions(): positions {}
+handle_event(event)
+forward(gameProgress, Player)
+ backward(GameProgression, Player)
+nextPlayer()
PlayerManager
+ current_player: Player
+add_player()
+next_player()
+get_current_player()
+getOtherPlayers()
+remove_current_player()
+get_length()
<<abstract>>
AnimalFactory
- createNew(type: AnimalType)
<<Enum>>
Colours
+WHITE
+BLACK
+GREY
StartPage
+players
+clear_screen()
+get_player_name()
+drawHome()
+initialiseGame(self,playerQuantity)

Sprint Contribution Log

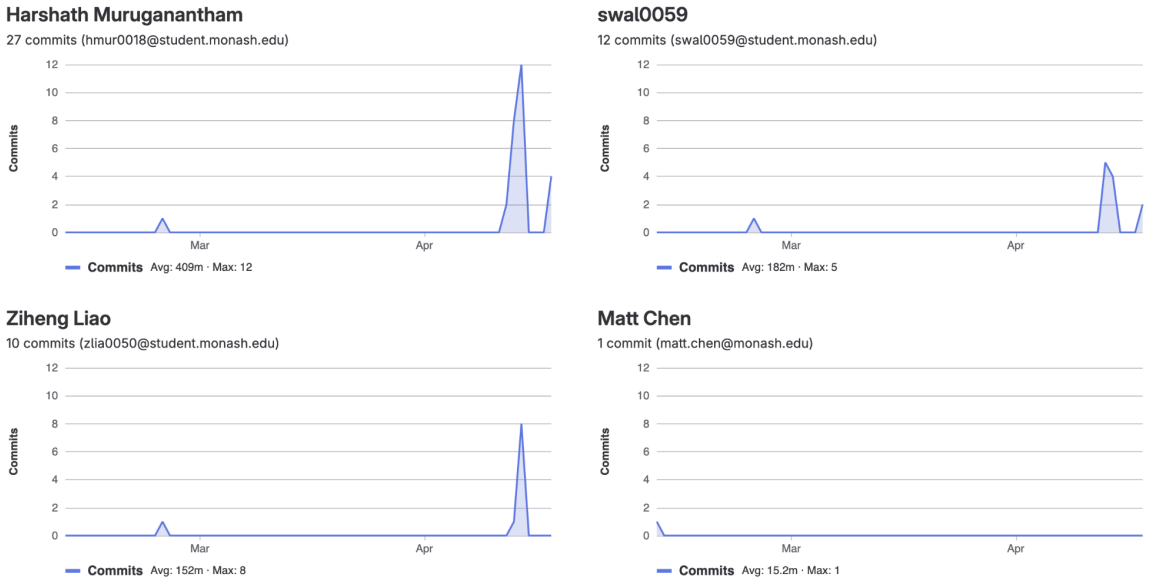
https://docs.google.com/spreadsheets/d/1HhfFLysPQRZ64nT7qyaP3tvSpjZD6gXlVTGKr04FE1Y/edit?usp=sharing

Task	Date Commenced	Student	Time Spent	Notes
Defining Metrics for Assessment Criteria - Functional Appropriateness	07/05/2024	Harshath Muruganantham	45mins	We attended and ran a group online meeting for 3 hours to define our criteria and metrics
Defining Metrics for Assessment Criteria - Maintainability	07/05/2024	Harshath Muruganantham	45mins	We attended and ran a group online meeting for 3 hours to define our criteria and metrics
Defining Metrics for Assessment Criteria - Availability	07/05/2024	Harshath Muruganantham	45mins	We attended and ran a group online meeting for 3 hours to define our criteria and metrics
Defining Metrics for Assessment Criteria - Functional Appropriateness	07/05/2024	Shannon Wallis	45mins	We attended and ran a group online meeting for 3 hours to define our criteria and metrics
Defining Metrics for Assessment Criteria - Maintainability	07/05/2024	Shannon Wallis	45mins	We attended and ran a group online meeting for 3 hours to define our criteria and metrics
Defining Metrics for Assessment Criteria - Availability	07/05/2024	Shannon Wallis	45mins	We attended and ran a group online meeting for 3 hours to define our criteria and metrics
Defining Metrics for Assessment Criteria - Functional Appropriateness	07/05/2024	Ziheng Liao	45mins	We attended and ran a group online meeting for 3 hours to define our criteria and metrics
Defining Metrics for Assessment Criteria - Maintainability	07/05/2024	Ziheng Liao	45mins	We attended and ran a group online meeting for 3 hours to define our criteria and metrics
Defining Metrics for Assessment Criteria - Availability	07/05/2024	Ziheng Liao	45mins	We attended and ran a group online meeting for 3 hours to define our criteria and metrics
Evaluating Shannon's Prototype against the criteria	08/05/2024	Harshath Muruganantham, Ziheng, Shannon	2 hours	We attended and ran an in person meeting for 3 hours to do the assessment of each of our prototypes
Evaluating Harshath's Prototype against the criteria	09/05/2024	Harshath Muruganantham, Ziheng, Shannon	2 hours	We attended and ran an in person meeting for 3 hours to do the assessment of each of our prototypes

Evaluating Ziheng's Prototype against the criteria	10/05/2024	Harshath Muruganantham, Ziheng, Shannon	2 hours	We attended and ran an in person meeting for 3 hours to do the assessment of each of our prototypes
Consolidated codebases of Harshath, Ziheng and Shannon	12/05/2024	Harshath Muruganantham	2 hours	Worked on Implementing the Game Instance
Worked on the class diagram	12/05/2024	Shannon Wallis, Harshath, Ziheng	2 hours	Worked on thinking through each of our own class diagrams and how the design will look for our consolodated approach
Added Flipping of Chit Cards and connected it to movements	12/05/2024	Harshath Muruganantham	2 hours	Worked on Implementing the Game Instance
Started implementation of Start Page Layout	12/05/2024	Harshath Muruganantham	1 hour	Worked on Implementing the Game Instance
Added Changing of Player turn	12/05/2024	Harshath Muruganantham	1 hour	Worked on Implementing the Game Instance
Addded Winning Game Funtionality	13/05/2024	Harshath Muruganantham	1 hour	Worked on Implementing the Game Instance
Added "edge case" code to stop players' turn when they "overshoot" their cave	13/05/2024	Harshath Muruganantham	1 hour	Worked on Implementing the Game Instance
Added the coding implementation of the start page	13/05/2024	Shannon Wallis	2 hours	Created the startpage that the user first sees
Added the coding implementation of display of player name against their respective token	13/05/2024	Shannon Wallis	1.5 hours	Coded up the functionality of displaying player name attached to the token on the top left of the screen
Touched up doc strings	14/05/2024	Ziheng Liao	20mins	Added doc strings to files that didn't already have it
Added class diagram components	14/05/2024	Shannon Wallis, Ziheng, Harshath	30mins	Added class diagrams onto lucid charts
Made a start on the CRC	14/05/2024	Shannon Wallis, Ziheng	1 hour	Added CRC cards into the Google Doc

Finished CRC Cards	16/05/2024	Ziheng Liao	1 hour	Added CRC cards into the Google Doc
Made the video	18/05/2024	Shannon, Ziheng, Harshath	1 hour	We ran an hour zoom to work on making a youtube video. We added the link to the pdf and read me
Worked on the finishing touches to the pdf	18/05/2024	Shannon, Ziheng, Harshath	1 hour	We looked back at the crc and redid parts of it
Submission	18/05/2024	Shannon, Ziheng, Harshath	20 mins	

Sprint Contribution Analytics



References

- [1]ISO25000, “ISO 25010,” *Iso25000.com*, 2019. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010
- P. K. Goyal and G. Joshi, "QMOOD metric sets to assess quality of Java program," 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), Ghaziabad, India, 2014, pp. 520-533, doi: 10.1109/ICICICT.2014.6781337. keywords: {Computational modelling;Java;Measurement;ISO;Encapsulation;Couplings;Java Programming language;QMOOD metrics;QMOOD model;software quality;software engineering},