# FIT3077 Sprint 1

## Team Name and Team Photo

Team Refactor


## Team Membership

**Fahad Assadi**
Contact Details: fass0001@student.monash.edu
Technical Strengths: OOP, Database Design, Css, Html, Java Script
Professional Strengths: Communication, Leadership, Time Management
Fun Fact: I can play the guitar relatively well

**Ahmad Ikram**
Contact Details: aikr0002@student.monash.edu
Technical Strengths: Java, Python, OOP
Professional Strengths: Problem-solving, Communication Skills, Attention to Detail
Fun Fact: I can ride a skateboard

**Umair Mohammad**
Contact Details: umoh0005@student.monash.edu
Technical Strengths: OOP, UI design
Professional Strengths: teamwork, team leadership and creativity
Fun Fact: I can move my ears without moving my face

**Hanif Mohammad Asif**
Contact Details: hmoh0035@student.monash.edu
Technical Strengths: Java, object-oriented programming
Professional Strengths: Teamwork, problem-solving
Fun Fact: I can move eyebrows individually

## Team Schedule

Weekly Monday meetings

| Date | Attendance | Agenda | Meeting Notes |
|---|---|---|---|
| 11/03/24 11pm | Fahad, Hanif, Umair | Allocating Sprint 1 sections | Allocated Team Information to Ahmad Ikram, User Stories to Umair, Domain Model to Fahad, Basic UI Design to Hanif |
| 18/03/24 4pm | Fahad, Hanif, Umair, Ahmad | Domain modelling and UI design | Learned the board game from the previous workshop and went through the game. Reviewed Domain modelling and discussed different models. |
| 25/03/24 5pm | Fahad, Hanif, Umair, Ahmad | Comparing Python and Java and also filled out Team Information | Decided on Java with JavaFX because it seemed more robust and better for building software |
| 26/03/24 4pm | Fahad, Hanif, Umair, Ahmad | Low-fi UI Design | UI design was discussed and new drawings were drawn up for the submission. |
| 27/03/24 4pm | Fahad, Hanif, Umair, Ahmad | Finalised all aspects of the assignment. | Review of assignment submissions and agreement on the assignment being ready for submission. |

## Team Justification

**Java**: While Python and Pygame offer viable alternatives for game development, our team's current expertise lies in Java. We chose Java due to its widespread use in software development, especially in enterprise applications. Java offers robust support for object-oriented programming, which aligns well with the structure of our board game project, and our team feels that it is better for software applications because Python is more suited for scripting/data use cases. Moreover, Java's cross-platform compatibility ensures that our game can run on various operating systems without significant modifications and easily create an executable.

**JavaFX**: JavaFX provides a modern, rich set of tools and APIs for building interactive UIs. Its integration with Java makes it a natural choice for our project, allowing us to create visually appealing and responsive interfaces. Since JavaFX is part of the Java Development Kit (JDK), it offers seamless integration with Java, simplifying development and deployment processes. Additionally, JavaFX's scene graph-based approach enables us to easily create complex UI layouts, facilitating the implementation of our board game's interface.

**Cross-Platform Compatibility**: JavaFX applications can be deployed across multiple platforms, including Windows, macOS, and Linux. This ensures that your board game will be accessible to a broader audience, regardless of their operating system. Furthermore, Java's "write once, run anywhere" (WORA) principle means that the codebase can be easily ported to different platforms without significant modifications, streamlining the deployment process.

**Community Support**: JavaFX benefits from a large and active community of developers who contribute to its ongoing development and provide support through forums, tutorials, and documentation. This extensive support network can be valuable for troubleshooting issues, seeking guidance on best practices, and accessing resources to enhance your game development process.

**Executable Creation and Distribution**: Java allows for creating standalone executable JAR files, which can be easily distributed and executed on other computers without requiring the same development environment. This ensures that others can share and play your board game without hassle, facilitating testing and feedback from external users.

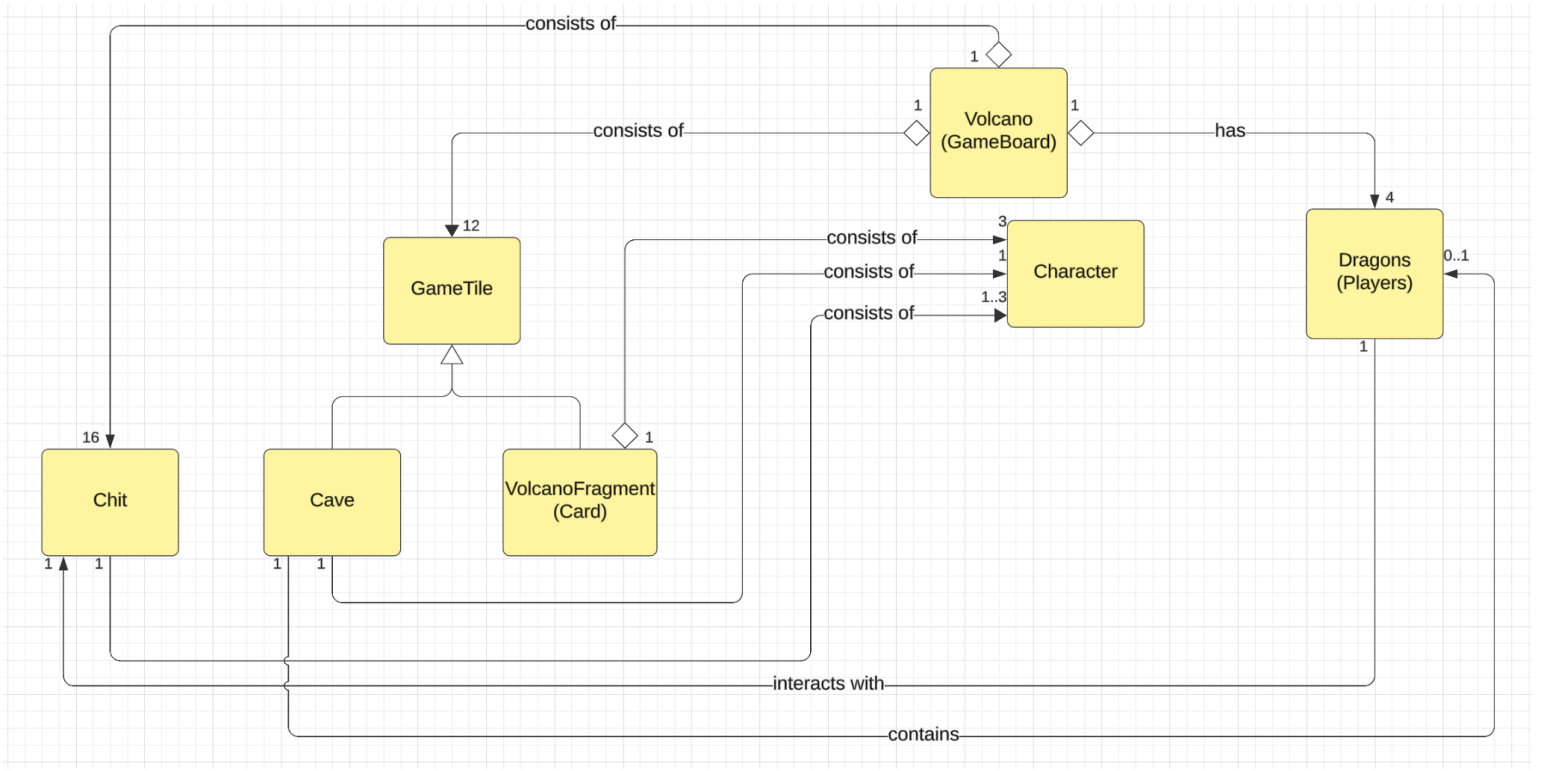
Overall, by choosing Java with JavaFX for your board game development, your team capitalises on its existing skills, benefits from a robust UI framework, ensures cross-platform compatibility, taps into a supportive community, and simplifies the executable creation and distribution process, ultimately enhancing the efficiency and success of your project.

## User Stories

1. As a player, I wish to select a dragon token at the beginning of the game to visually represent myself on the game board.
2. As a player, I want to see a visual representation of the volcano and all players' dragons on the game interface, so I can understand the game's current state.
3. As a game, I want to shuffle the 16 dragon chits and place them face down within the volcano area at the start of the game, ensuring a fair and random beginning.
4. As a developer, I need to enable dragon chit interaction through mouse clicks, allowing players to easily use and engage with chits during gameplay.
5. As a chit, I need to only allow the dragon to move if the drawn chit matches the creature on the player's current square, enforcing game rules.
6. As a chit, I want to automatically move the player's dragon around the board based on the creature shown on the chit I draw, adhering to the game rules.
7. As a player, I want the game to notify me when it's my turn so that I know when to make my move.
8. As a dragon pirate chit, I need to automatically move a player's dragon backwards when they encounter a dragon pirate chit, enforcing the game rules.
9. As a chit, I want to automatically revert to a hidden state after being revealed, ensuring the game's challenge is maintained.
10. As a game, I must prevent more than one dragon from occupying the same square, to maintain fairness and adhere to the game's rules.
11. As a game, I need to ensure that a dragon can only enter its cave with the exact number of moves required, preserving the integrity of the game's end condition.
12. As a player, I want the game to declare the winner once a dragon successfully navigates back to its cave, marking the end of the game.
13. As a player, I want the ability to restart the game once finished, so I can play multiple rounds without exiting the application.
14. As a developer, I want to provide clear, concise documentation on how to set up and run the game, ensuring it is accessible for users on different platforms.
15. As a player, I would like to see animations for dragon movements, making the game more visually appealing and engaging.
16. As a player, I'm interested in a feature that explains the rules and offers strategy tips, making the game more accessible to new players.
17. As a player, I want to see a leaderboard at the end of the game, so I know how I ranked against other players.
18. As a developer, I need to ensure the game is playable on a single device without server-side code, following project requirements.
19. As a game, I want to automatically save the game progress, so players can resume their game even after closing the application.
20. Extension: As a developer, I want to implement an AI opponent for single-player mode, offering a challenging experience even without human opponents.
21. Extension: As a player, I want to use a 'Game dynamic chits' like player position swap chit or reverse direction chit to add an unexpected twist to the game's progression.
22. Extension: As a player, I want to use a 'Point Multiplier Power up' chit to double the moves I earn in a turn, allowing me to advance quicker towards my cave.

23. Extension: As a player, I want to utilise a 'Shove Back' chit to force all other players to move back a specified number of spots, giving me a competitive edge.
24. Extension: As a system, when two dragons land on the same square, I want to initiate a 'Tic-Tac-Toe' mini-game to determine who gets to stay on the square, adding a layer of skill-based challenge.

## First Draft




The first draft massively misrepresented the concept of a domain model. It doesn't include all the entities involved in the problem space. Entities like the different characters (baby dragon, dragon pirate, spider, bat) were completely omitted even though they play a significant role in the game space due to the fact that these entities can be represented as instantiations of the character class (misunderstanding between uml diagrams and domain models). The cardinalities between different entities were hardcoded under the false pretence that the game would never have a situation where these cardinalities would change.

## Entity Explanations:

**Volcano (GameBoard):** The gameboard that all entities exist on. The players interact with all game objects through the game board. The GameTiles, Characters, Chits all exist on the game board.

**GameTile:** An abstraction for the entities that the player entity can walk on. This entity forms the basis for what the Cave entity and Volcano Fragment entity are.

**Chit:** A pickable entity that exists in the centre of the gameboard that determines the amount of steps that the player moves. They are the only entity that the player can interact with.

**Cave:** The starting and ending game tiles for the players. The win condition for a player is to start at a cave and end at the same cave.

**VolcanoFragment:** The tiles that the player will have to traverse to reach the ending cave. They contain 3 characters that determine if the player can move forward the current turn.

**Dragons (Players):** The player entity of the game that users will use to determine their current standing in the game.

**Character:** An abstraction for the different entities (baby dragon, dragon pirate, spider, bat) that appear on the game board on chits, caves and volcano fragments.

# Entity Relation Explanations:

## Generalisation:

GameTile generalises Cave and Volcano Fragment entities as they are both entities that the world can generalise to objects that the player can traverse on.

## Association:

Players interact with chits to be able to traverse forward or backward through the game.

Every Chit is made up of a character entity that defines its interactions with other entities. So the chits have associations with characters.
Similar to the previous association, caves also contain character and player entities in the problem space.
(We eventually figure out that these were a mistake to have as an association)

Caves house the player during the start and the end of the game but since these entities can exist independently of each other, they have an association relationship.

## Aggregation:

Volcano Fragment forms an aggregation with Character as every volcano fragment needs 3 characters to exist.

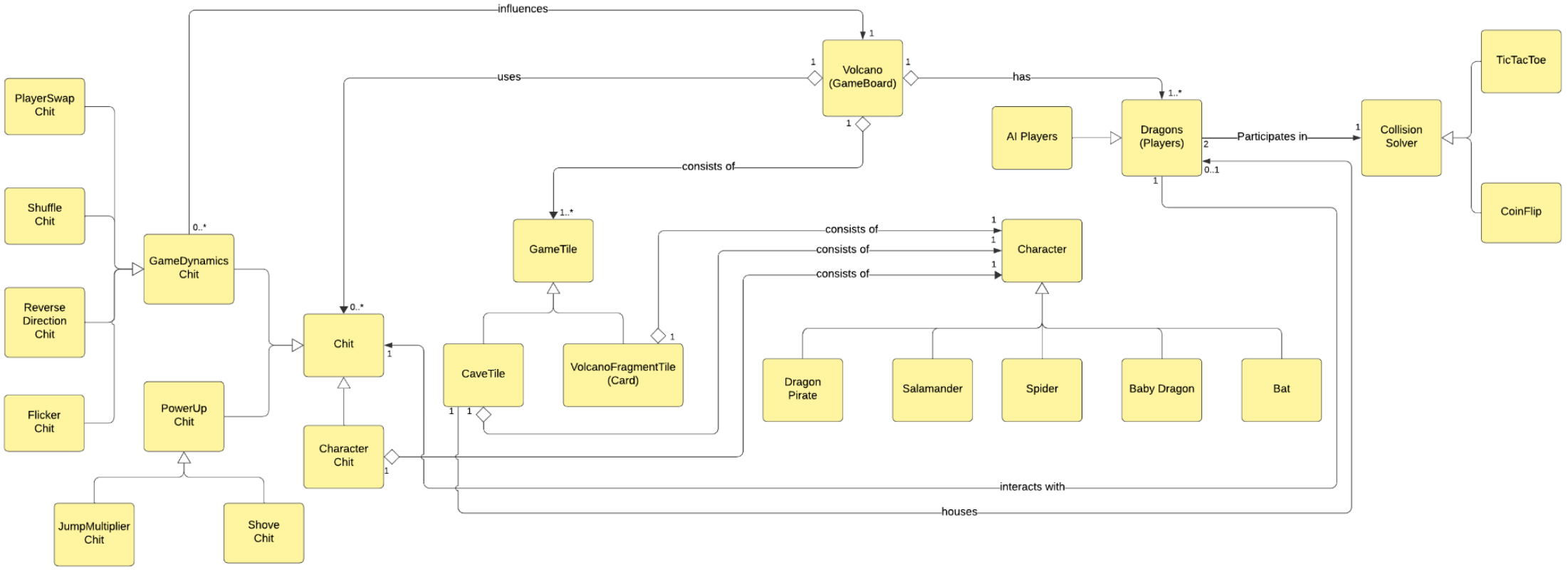
The game board is built up of the game tile, player and chit entities so these will all form aggregation with the game board.

## Cardinalities:

All cardinalities in the first draft are hardcoded values based on the base game. In the base game there are 8 cards and 4 cave tiles that the player can traverse across causing the cardinality between GameTile and gameboard to be 12 to 1. There are 16 chits that are made up of character entities. The Gameboard also has 4 players at all times. All these lead to their respective cardinalities to 1 gameboard. The cave can house a single player during the start or end of a game leading to a 0 or 1 cardinality between them. Caves can also consist of a character that determines the start of a game leading to a 1 to 1 cardinality.

Players also interact with chits at a 1 to 1 cardinality as a player can only interact with 1 chit at a time.

# Final Draft:


influences
uses
Volcano (GameBoard)
has
consists of
PlayerSwap Chit
Shuffle Chit
Reverse Direction Chit
Flicker Chit
GameDynamics Chit
PowerUp Chit
JumpMultiplier Chit
Shove Chit
Chit
Character Chit
GameTile
CaveTile
VolcanoFragmentTile (Card)
consists of
consists of
consists of
Character
Dragon Pirate
Salamander
Spider
Baby Dragon
Bat
AI Players
Dragons (Players)
Participates in
Collision Solver
TicTacToe
CoinFlip
interacts with
houses

The final version of the domain model consists of all the changes made over weeks of understanding the problem space and the right way to model domains. It contains changes like including all the entities involved in the problem space, changing the cardinalities to reflect a constantly evolving system, and adding all the extensions we plan to add to the game. For our extension, we decided to add more types of chits. We came up with three kinds of game dynamic chits that will change the game dynamics, powerup chits that will help the player win, and the character chits from the default game. Furthermore, we have an option for an AI player, and we also added a collision system where if you land on a tile with a player already on it you can battle for that position with a tic-tac-toe or coin flip.

## Entity Explanations:

**Volcano (GameBoard):** The gameboard on which all entities exist. The players interact with all game objects through the game board. The GameTiles, Characters, Chits all exist on the game board.

**GameTile:** An abstraction for the entities that the player entity can walk on. This entity forms the basis for what the Cave entity and Volcano Fragment entity are.

**Chit:** An entity that exists in the centre of the gameboard that determines the amount of steps that the player moves. They are the only entity that the player can interact with.

**CaveTile:** The starting and ending game tiles for the players. The win condition for a player is to start at a cave and end at the same cave.

**VolcanoFragment(Tile):** The tiles that the player will have to traverse to reach the ending cave.

**Dragons (Players):** The player entity of the game that users will use to determine their current standing in the game.

**AI Players [Extension]:** AI entity that can be a replacement to Dragons (Players). It is meant to play similarly to real players in the absence of real players.

**Character:** An abstraction for the different entities (baby dragon, dragon pirate, spider, bat) that appear on the game board on chits, caves and volcano fragments.

**Baby Dragon:** A character entity that is present in tiles and chits.

**Salamander:** A character entity that is present in tiles and chits.

**DragonPirate:** A character entity that is present in tiles and chits.

**Bat:** A character entity that is present in tiles and chits.

**Spider:** A character entity that is present in tiles and chits.

**CharacterChit:** A chit which indicates the number of moves a player can move corresponding to a character. In the base game, every chit is a character chit, but with extended gameplay mechanics, there are chits with no associated characters for power ups, game dynamic changes etc.

**PowerUpChit [Extension]:** An abstraction that allows players to stumble onto one of the power up chits that will affect their gameplay.

**JumpMultiplier[Extension]:** A power up chit which multiplies the number of steps that is taken by a player when the player picks a character chit.

**ShoveChit [Extension]:** A power up chit which pushes back enemy players a certain number of tiles.

**GameDynamicsChit [Extension]:** The type of chit that vastly affects the game dynamics for every player.

**PlayerSwapChit [Extension]:** This is a type of game dynamic chit which swaps positions of players on the board.

**ShuffleChit [Extension]:** This is a type of game dynamic chit which is in charge of shuffling the chits between play periods.

**ReverseDirectionChit [Extension]:** This is a type of game dynamic chit that reverses the direction of movement for every player.

**FlickerChit [Extension]:** This is a type of game dynamic chit that flickers and reveals all the chits on the board momentarily.

**CollisionSolver [Extension]:** An abstraction for the mechanism that manages what happens when two players end up on the same tile.

**TicTacToe [Extension]:** A collision solver that allows the two players having a collision to play a game of TicTacToe to determine who gets to be on the contested tile.

**CoinFlip [Extension]:** A collision solver that uses a coin flip to determine who gets to be on the contested tile.

# Entity Relation Explanations:

## Generalisation:

Chit is a generalisation for all the different types of chits that a player can interact with, these include other generalisations for subsets of chits. We have Character chits, GameDynamics Chits and the PowerUp Chits, that are all generalised by chits.

The GameDynamics Chits are further subdivided into different types of entities that can affect the game, these include the PlayerSwap Chit, the Shuffle Chit, the ReverseDirection Chit and the FlickerChit that all have functionalities that can vastly affect the game dynamics allowing them to be generalised by the GameDynamics Chit Entity.
The PowerUp Chits are further subdivided into different types of chit entities that can give the player certain advantages based on the chit they picked. These advantages include the JumpMultiplier Chit and the Shove Chit. Since these entities all have the common functionality to give the player interacting with this chit an unfair advantage, they can be generalised by the PowerUp Chit.

GameTile generalises Cave and Volcano Fragment entities as they are both entities that the world can generalise to objects that the player can traverse on.

Character is a generalisation for all the different types of characters like the baby dragon, dragon pirate, spider and bat.

Players is a generalisation for the AI player as AI players are players with the added feature of making moves similar to real players. These can be generalised under the player entity.

Collision solver is an abstraction for all the different ways a collision can be solved when a collision occurs. Currently there are two ways to solve a collision, the TicTacToe and the CoinFlip. Since these can be defined under the umbrella term of collision solvers, they can be generalised by collision solvers.

## Association:

Players interact with chits to be able to traverse forward or backward through the game leading to an association relationship.

Caves house the player during the start and the end of the game but since these entities can exist independently of each other, they have an association relationship.

GameDynamics Chits influence the way the game is played in various ways but these two entities can exist independently of each other leading there to be an association between them.

Players interact with the Collision Solver entity whenever two players happen to land on the same game tile. This leads to player and collision solver entities having an association relationship.

## Aggregation:

The game board is built up of the game tile, player and chit entities so these will all form aggregation with the game board as the game board cannot exist independently with any of these entities missing, they form the basis of the game.

Character Chits, Volcano Fragments and CaveTile can only exist when they have a character associated with them. They cannot exist independently of the character. Thus they all form aggregations with the character entity.

### Cardinalities:

A VolcanoFragmentTile card is associated with characters through a one-to-one aggregation relationship. This means one fragment tile can contain one character.

Each CaveTile is linked to a character through a one-to-one relationship, indicating that every cave is designed to accommodate no more than one character at any given time.

The Character Chit is in a one-to-one relationship with a character. This signifies that every character is uniquely paired with a single Character Chit, underscoring the exclusivity of this association.

The relationship between Dragon (players) and the collision solver is two-to-one. This particular cardinality reflects that when collisions occur, they involve two "Dragons" being resolved by a single collision solver.

The Volcano (Gameboard) has a one-to-many aggregate relationship with Dragons(Players), GameTile, and Chit.This arrangement illustrates that the Volcano game board is composed of these entities, encompassing multiple Dragons (Players), GameTiles, and Chits as part of its structure.

GameDynamicsChit has a many to one relationship with Volcano (Gameboard) since there are many game dynamic chits and they can all influence the single Volcano.

## Specific Choices:

We chose to implement the Chit entity, which can be inherited by CharacterChit. Even though the base game has chits that can only contain the predefined characters, we chose to extend the functionality of chits by including mechanics like powerups where no character is associated with the chit. This choice was made so that the game can be extended in interesting ways.

We chose to implement a Game Dynamics Chit, which could alter the way the game is played by all the players once chosen. This was done so that the game has greater longevity and player retention. This chit is itself an abstraction for any chit which can alter game dynamics such as Shuffle Chit and Reverse Direction Chit. This was designed this way so that more gameplay altering mechanics may be added in the future.

Collision Solver is an entity that was modelled to allow for interesting gameplay mechanics to be paired with player collisions. This entity is an abstraction of any other entity which performs collision solving, such as the tic-tac-toe minigame or the coin flip minigame. The modelling was done this way to allow for further minigames to be introduced if desired as collision solvers.

## Assumptions:

Some assumptions made for our model are that currently, there are only four types of Game tiles (Salamander, Spider, Baby Dragon and Bat), there is no feature for extending any more characters, and as a consequence, there will be only four caves. Further assumptions that were made were entity relations that were left out that we thought were more UML implementation type of relationships for example, we left out a relationship between collision solver and game tile for simplicity.

## Contribution Analytics

# Contribution Log

## Contribution Log

Contributions:

| Date | Name | Task | Time Spent | Description |
|---|---|---|---|---|
| 14/03/24 | Umair | Team Information | 0.5 hrs | Completed Team Name and Team Photo and Team Membership |
| 15/03/24 | Ahmad | Basic UI Design | 1.5 hrs | Redesigned UI with more gameplay details |
| 16/03/24 | Hanif | User Stories | 0.45 hrs | Added User Stories for gameplay initialization |
| 16/03/24 | Fahad | Domain Model and Justification | 2 hrs | Added the first draft of the domain model and the justification for the domain model, including explanations for all entities within the domain model. |
| 17/03/24 | Ahmad | Team Information | 0.5 hrs | Completed Team Schedule |
| 18/03/24 | Fahad | Basic UI Design 2nd | 0.5 hrs | Added more gameplay sequences |
| 19/03/24 | Ahmad | User Stories | 0.5 hrs | Added user stories relating to relating to player, chits, players blocking |
| 20/03/24 | Umair | Domain Model 2nd | 1 hrs | Added variable number of players, tiles, chits, chit shuffler |
| 21/03/24 | Hanif | Team Information | 1 hrs | Justified java and javafx for the tech stack |
| 22/03/24 | Hanif | UI Design 3nd | 0.75 hrs | Added more gameplay sequences |
| 22/03/24 | Umair | User Stories | 0.75 hrs | Added 6 User Stories |
| 24/03/24 | Fahad | Team Information 4th | 1 hrs | Justified Cross-Platform Compatibility, Community Support and Executable Creation and Distribution |
| 24/03/24 | Umair | Basic UI Design | 0.25 hrs | Finalized UI Design |
| 24/03/24 | Hanif | Domain Model | 0.5 hrs | Added various Game Dynamics Chits to the domain model |
| 26/03/24 | Fahad | User Stories 4th | 0.5 hrs | Added 6 User stories |
| 27/03/24 | Ahmad | Domain Model | 1 hrs | Added Collision Solver mechanic with tictactoe and coinn flip |
| 27/03/24 | Fahad | Final Domain Model | 2 hrs | Updated and refined the domain model to have all the final changes |