MONASH University

FIT3077 Software Engineering: Architecture and Design
Sprint Four Submission

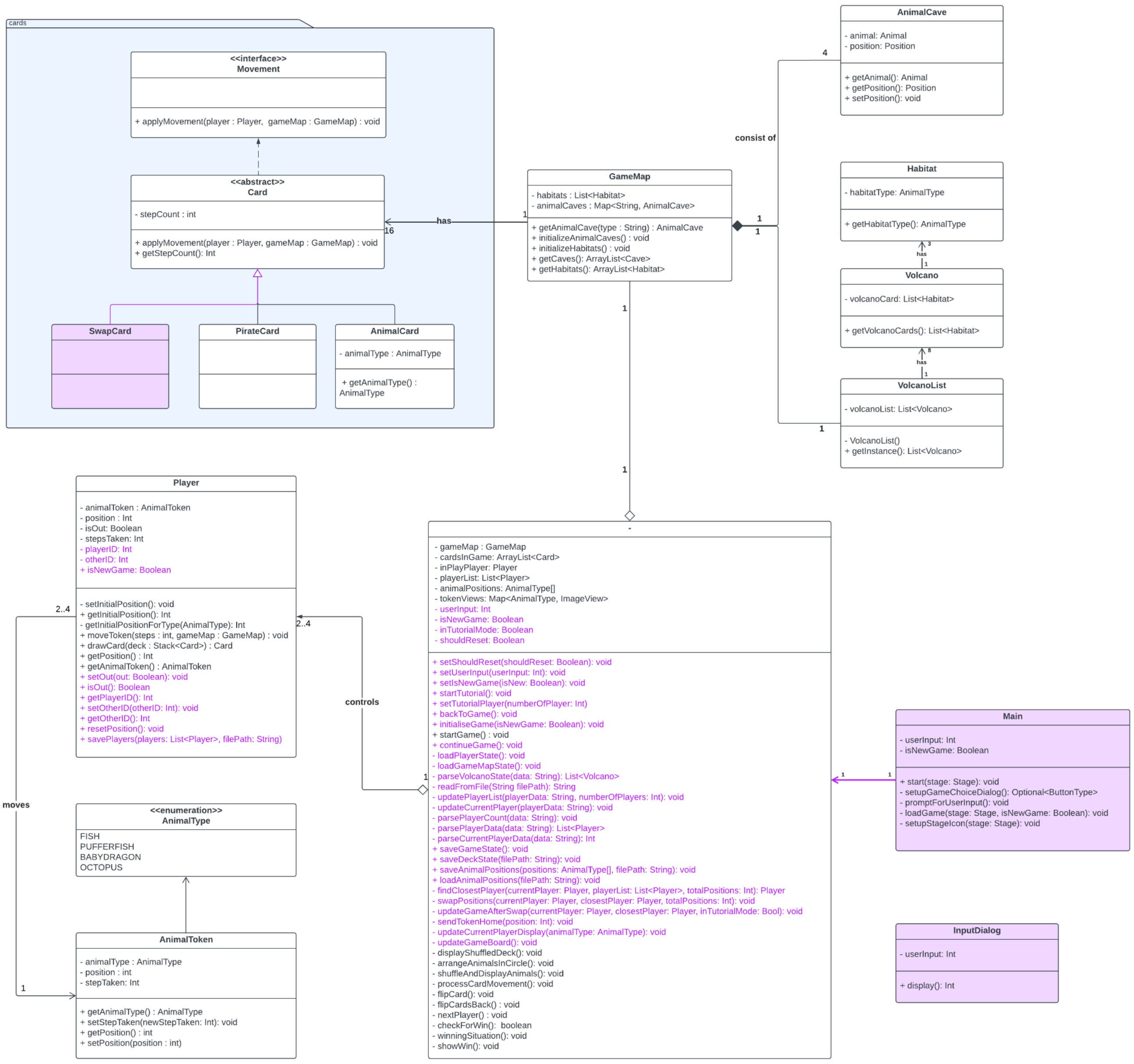
CL_Monday06pm_Team988

Group Report

Group Members:	Email Address:
Sheena Quah Xin Yee	squa0013@student.monash.edu
Samantha Oh Jia-Jia	sohh0008@student.monash.edu
Racheal Lim	rlim0050@student.monash.edu
Ng Jeh Guan	jngg0104@student.monash.edu

Table of Contents

1.0 Object Oriented Design


cards
<<interface>>
Movement
+ applyMovement(player : Player, gameMap : GameMap) : void
<<abstract>>
Card
- stepCount : int
+ applyMovement(player : Player, gameMap : GameMap) : void
+ getStepCount(): Int
SwapCard
PirateCard
AnimalCard
- animalType : AnimalType
+ getAnimalType() : AnimalType
has
16
1
GameMap
- habitats : List<Habitat>
- animalCaves : Map<String, AnimalCave>
+ getAnimalCave(type : String) : AnimalCave
+ initializeAnimalCaves() : void
+ initializeHabitats() : void
+ getCaves(): ArrayList<Cave>
+ getHabitats(): ArrayList<Habitat>
consist of
4
1
1
1
AnimalCave
- animal: Animal
- position: Position
+ getAnimal(): Animal
+ getPosition(): Position
+ setPosition(): void
Habitat
- habitatType: AnimalType
+ getHabitatType(): AnimalType
3
has
1
Volcano
- volcanoCard: List<Habitat>
+ getVolcanoCards(): List<Habitat>
8
has
1
VolcanoList
- volcanoList: List<Volcano>
- VolcanoList()
+ getInstance(): List<Volcano>
1
1
Player
- animalToken : AnimalToken
- position : Int
- isOut: Boolean
- stepsTaken: Int
- playerID: Int
- otherID: Int
+ isNewGame: Boolean
- setInitialPosition(): void
+ getInitialPosition(): Int
- getInitialPositionForType(AnimalType): Int
+ moveToken(steps : int, gameMap : GameMap) : void
+ drawCard(deck : Stack<Card>) : Card
+ getPosition() : Int
+ getAnimalToken() : AnimalToken
+ setOut(out: Boolean): void
+ isOut(): Boolean
+ getPlayerID(): Int
+ setOtherID(otherID: Int): void
+ getOtherID(): Int
+ resetPosition(): void
+ savePlayers(players: List<Player>, filePath: String)
2..4
2..4
controls
1
moves
1
<<enumeration>>
AnimalType
FISH
PUFFERFISH
BABYDRAGON
OCTOPUS
AnimalToken
- animalType : AnimalType
- position : int
- stepTaken: Int
+ getAnimalType() : AnimalType
+ setStepTaken(newStepTaken: Int): void
+ getPosition() : int
+ setPosition(position : int)
-
- gameMap : GameMap
- cardsInGame: ArrayList<Card>
- inPlayPlayer: Player
- playerList: List<Player>
- animalPositions: AnimalType[]
- tokenViews: Map<AnimalType, ImageView>
- userInput: Int
- isNewGame: Boolean
- inTutorialMode: Boolean
- shouldReset: Boolean
+ setShouldReset(shouldReset: Boolean): void
+ setUserInput(userInput: Int): void
+ setIsNewGame(isNew: Boolean): void
+ startTutorial(): void
+ setTutorialPlayer(numberOfPlayer: Int)
+ backToGame(): void
+ initialiseGame(isNewGame: Boolean): void
+ startGame() : void
+ continueGame(): void
- loadPlayerState(): void
- loadGameMapState(): void
- parseVolcanoState(data: String): List<Volcano>
- readFromFile(String filePath): String
- updatePlayerList(playerData: String, numberOfPlayers: Int): void
- updateCurrentPlayer(playerData: String): void
- parsePlayerCount(data: String): void
- parsePlayerData(data: String): List<Player>
- parseCurrentPlayerData(data: String): Int
+ saveGameState(): void
+ saveDeckState(filePath: String): void
+ saveAnimalPositions(positions: AnimalType[], filePath: String): void
+ loadAnimalPositions(filePath: String): void
- findClosestPlayer(currentPlayer: Player, playerList: List<Player>, totalPositions: Int): Player
- swapPositions(currentPlayer: Player, closestPlayer: Player, totalPositions: Int): void
- updateGameAfterSwap(currentPlayer: Player, closestPlayer: Player, inTutorialMode: Bool): void
- sendTokenHome(position: Int): void
- updateCurrentPlayerDisplay(animalType: AnimalType): void
- updateGameBoard(): void
- displayShuffledDeck(): void
- arrangeAnimalsInCircle(): void
- shuffleAndDisplayAnimals(): void
- processCardMovement(): void
- flipCard(): void
- flipCardsBack() : void
- nextPlayer() : void
- checkForWin(): boolean
- winningSituation(): void
- showWin(): void
1
1
Main
- userInput: Int
- isNewGame: Boolean
+ start(stage: Stage): void
- setupGameChoiceDialog(): Optional<ButtonType>
- promptForUserInput(): void
- loadGame(stage: Stage, isNewGame: Boolean): void
- setupStageIcon(stage: Stage): void
InputDialog
- userInput: Int
+ display(): Int

Note: Those attributes, methods, relationships and classes highlighted in purple are newly added for Sprint4.

2.0 Reflection on Sprint 3 Design

2.1 Saving and Loading A Game

Level of Difficulty: Easy-moderate

When implementing the save game functionality, we had to record the player's current location. Upon quitting the game, some players might still be in their respective caves, while some might already be outside of their caves (i.e. in some habitat on the game board). To track this aspect of their locations, we added a boolean isOut in the Player class. Therefore, when saving the player's state, we pass a boolean as a parameter to the savePlayers() function which is responsible for writing each player state into player_list.txt. This boolean is then used in the updateGameBoard() function, where it first checks if the player is out of their cave. If not, the player's token is displayed such that it is in their cave (i.e. their initial position). The main challenge was to ensure that this boolean is correctly saved and loaded. Integrating this with the game board display logic also added some complexity but was manageable.

Another boolean added to the Player class was the isNewGame boolean, which records if the current game state is a completely new game (true) or the game is loaded from a previously saved state (false). If the current game is loaded from a previously saved state, set the token's isOut property to whatever is defined. This boolean is also added to the GameController class, which is used to conduct several checks in the startGame() function. For example, if a new game is started, we would want to reset all game states such as the player list and the in-play player. Else, the previously saved game state will be loaded and displayed. This integration was relatively easy to implement as the existing implementation for starting a game facilitates this change without significant issues.

What could be improved was we could have included the use of player positions (index) on the map in our last sprint. We did store player positions as a class property, but we have never used it in any part of the game. Even for determining a winning situation, we only took the player's total step taken into consideration. However, as we implemented this functionality, we realised that the total steps taken by a player gives insufficient information when saving/loading game data. Thus, we integrated the use of player positions, which allows us to know where the players' tokens should be displayed on the board when loading a saved game.

Overall, implementing this functionality was at moderate difficulty. Our Sprint 3 design/implementation made saving the game easier than expected. In our previous implementation, we defined properties for various classes representing game components such as player ID, volcano card instances, and card type which made saving the game state rather straightforward. With just minor changes to the mentioned classes, we only had to write the relevant properties that represent the current game state into the text file. Furthermore, loading a saved game can be done by parsing the previously saved game data from the relevant text files. The parsed data can then be used to create game elements directly with the use of existing class constructors

2.2 Swap Card

Assumption for this extension
(1) The current player animal token has to go one more round if it has been swapped past its cave.
(2) If a person switches places with another player before them, they will get to the cave sooner and have a better chance of winning.

Level of Difficulty: Moderate

During Sprint 3, we implemented positional tracking for 26 habitats, excluding caves, because only a few simple actions were needed to move tokens into and out of caverns. Thus, we decided to simplify our design by not keeping cave placements. We had an attribute defined in the Player class, which is a boolean variable, isOut, to track whether a token was inside its cave. Because of this configuration, it was rather easy to build a feature that allows a token to swap with the closest token on the game map. Tokens that were still in their caves could be ignored, and the closest player could be quickly found by some calculation. In addition, Additionally, in our earlier implementation, we developed an enum class named CardType, which contains AnimalCard and PirateCard, to specify the sorts of "chit" cards. This allowed us to easily direct the tokens to move according to the type of card that was picked. As we had to add a new card for this extension, we can define another card type for it (SwapCard) in the enum class for straightforward integration. This card type functions similarly to animal and pirate cards in that the token will be ordered to swap if the controller detects that a swap card has been selected.

However, due to the mishandling of the cave locations, we found that it was challenging to swap the two tokens when the current token that was in play hadn't left its cave. In our previous implementation, each player could only enter their own cave. Our current method had to be changed in order to accommodate the new criteria, which stated that tokens were required to visit any cave during a swap. By expanding our location monitoring to incorporate cave positions in addition to the 16 habitats, the tokens now can access other caves when it's necessary.

Overall, integrating this extension was more challenging than we initially anticipated but still manageable. Our pre-existing game structure made it obvious to deal with the problem. We already had a clear mind and direction to fix the issues we mentioned above. One of our major changes is the requirement for including cave positions. In spite of this, the smooth integration of the new functionality was made possible by our codebase's low level of code smells. The significance of creating adaptable systems that can quickly adjust to new requirements was highlighted by this experience.

2.3 Self-Defined Extensions: Tutorial Mode

Level of Difficulty: Easy-Moderate

In the GameController class, we added a few new class properties and class functions to set up elements for the game tutorial view. Moreover, we added some FXML elements to the same class. These form the entire view pane of the tutorial screen. Similar to the isNewGame boolean, a boolean inTutorialMode is added to this class. This allows for checks in various functions such as startGame() and processCardMovement. These functions use this boolean to determine the setting up of the game board and update game labels. The game instructions label displays different text depending on the type of game mode when a card is flipped and a movement is to be processed.

Similar to setting up a new game, we set up all game elements on the game board (on the tutorial pane) including players. As for players, we added a boolean class property isStatic to the Player class. This determines if the player can move their token, and is used when creating players for the tutorial mode. For demonstration purposes, we only allow one player to move in the tutorial mode, thus this player has their isStatic property set to “true” while the remaining is set to “false”.

Overall, the implementation of this extension was at easy-to-moderate difficulty because it involved multiple checks throughout the GameController class and we got slightly confused during the development process. We came to realise that the majority of the game elements and functions required for this extended functionality was already implemented previously (e.g. setting up the game board and initial game state). No major changes to existing classes were required for this extended functionality.

2.4 Self-Defined Extensions: Send Opponent Token Home When Met

Level of Difficulty: Easy-Moderate

To make our game more challenging, our team decided to create a new rule - when the token is about to move to an occupied Tile, instead of being blocked, the token will “kick” the opponent token to its “home” cave. We want to create a new dynamic where players may essentially "reset" their opponents, which has a substantial influence on player interactions and game strategy. The modification made it possible for us to easily reset the position of each token to its cave by utilising the pre-existing properties that recorded each token's initial location. The previous configuration made it possible to adjust to the new demand with ease.

To implement this functionality, we first check if the target habitat contains a token. If an opponent’s token is found, it is reverted to its original location and sent back to its home cave. This required the person in control of the token to start again at their starting point. Subsequently, the current token could move to the target habitat, which is now vacant.

This new rule simply required minimal tweaks to our pre-existing infrastructure for token moves and placements, greatly reducing the complexity and implementation time and making it straightforward to reset the token's current position to its cave. Even though this extension is simple to apply, there are useful lessons to be learned for further work. The pre-existing design, which recorded each token's starting location, turned out to be quite helpful, proving how important thorough and forward-thinking planning is in the early phases of development. Nonetheless, the necessity for flexibility in design should not be overshadowed by this specific extension's simplicity of use. Going forward, future adjustments might be made even more efficiently by using design patterns that account for different game dynamics and possible rule changes.

2.5 Conclusion

By reflecting on our Sprint 3 and considering the level of difficulty in integrating additional features, we learned a lot about our design’s extensibility. Since every extension has its own set of difficulties and chances for development, the save and load process was simple due to the pre-existing game structure, but it did emphasise how crucial it was to include player positions right away. Our existing enums and structure made the switch card feature easier to integrate, even if it required tweaks to take into account cave locations. Even though it required several tests, tutorial mode profited from our earlier use of game setup logic. It was simple to implement the new rule about sending opponent tokens home since we had prearranged where the tokens would be placed.

Overall, while our Sprint 3 design was rather flexible, certain places may have been better planned out to make future modifications easier. If we were to start over, we would be able to boost the flexibility of our system by using design patterns and maintaining comprehensive documentation. This idea highlights the necessity of vision, meticulous preparation, and adaptable design patterns in software development to better handle future additions.