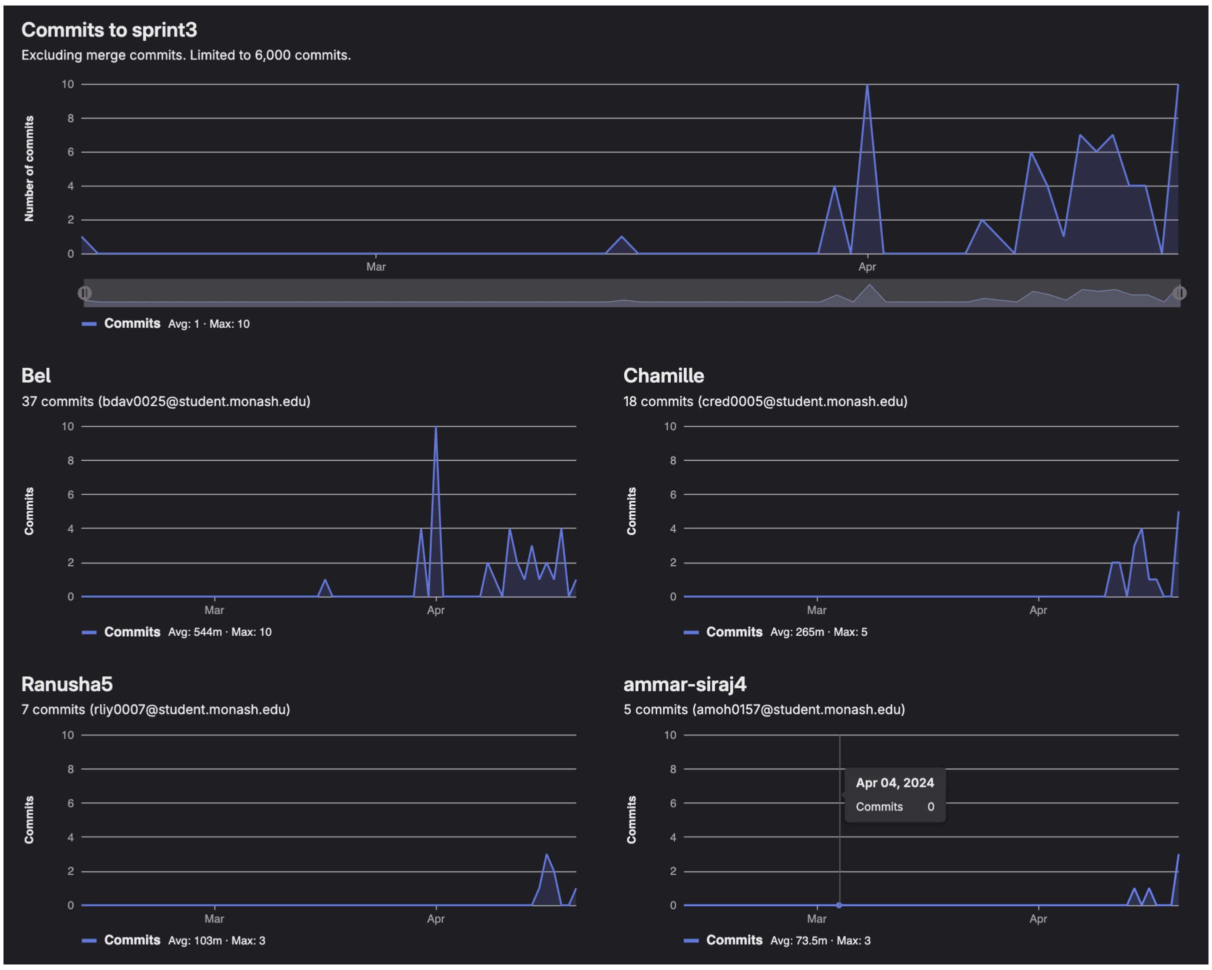
FIT3077 - Sprint 3 Deliverables

Group Members

Ranusha Liyanage
Chamille Reddiar
Belvinjeet Kaur
Mohamed Ammar Ahamed Siraj Mohamed

Contributor Analytics



Commits to sprint3
Excluding merge commits. Limited to 6,000 commits.

Commits Avg: 1 · Max: 10

Bel
37 commits (bdav0025@student.monash.edu)

Commits Avg: 544m · Max: 10

Chamille
18 commits (cred0005@student.monash.edu)

Commits Avg: 265m · Max: 5

Ranusha5
7 commits (rliy0007@student.monash.edu)

Commits Avg: 103m · Max: 3

ammar-siraj4
5 commits (amoh0157@student.monash.edu)

Apr 04, 2024
Commits 0

Commits Avg: 73.5m · Max: 3

Assessment criteria to evaluate the tech-based software prototypes

1. **Functional suitability:**
    - Functional completeness - the set of functions covers all the specified tasks and intended users' objectives.
    - Functional correctness - system provides accurate results when used by intended users.
    - Functional appropriateness - functions facilitate the accomplishment of specified tasks and objectives.

2. **Reliability**
    - Availability - product is operational and accessible when required for use.

3. **Maintainability**
    - Modularity - Degree to which a system is composed of discrete components such that a change to one component has minimal impact on other components.
    - Reusability - a product can be used as an asset in more than one system, or in building other assets.
    - Modifiability - Degree to which a system can be effectively and efficiently modified without introducing defects or degrading existing product quality.

4. **Flexibility**
    - Adaptability - system can effectively and efficiently be adapted for or transferred to different hardware software or other operational or usage environments.
    - Scalability - product can handle growing or shrinking workloads or to adapt its capacity to handle variability.
    - Installability - system can be successfully installed and/or uninstalled in a specified environment.

5. **Interaction capability**
    - Appropriateness recognizability - users can recognize whether a product or system is appropriate for their needs.
    - Learnability - system can be learnt to be used by specified users within a specified amount of time.
    - Operability - system has attributes that make it easy to operate and control.
    - User engagement - user interface presents functions and information in an inviting and motivating manner

Assessment of all Sprint 2 tech-based prototypes

Criteria assessment metrics: 1 is worst & 5 is best

Name:		Chamille	Bel	Ranusha	Ammar
Criteria:					
Functional suitability	Completeness	4	5	5	5
	Correctness	4	4	4	5
	Appropriateness	4	4	4	5
Maintainability	Modifiability	4	3	3	3
	Reusability	4	3	4	4
	Modularity	5	4	5	2
Flexibility	Adaptability	2	3	2	2
	Scalability	2	3	2	3
	Installability	1	5	3	5
Reliability	Availability	1	5	4	5
Interaction capability	Appropriateness recognizability	5	5	4	5
	Learnability	4	3	3	4
	Operability	3	4	4	4
	User engagement	4	4	4	3

For Chamille's prototype, the main classes were board and factories. Ideas/elements to be used in Sprint 3 are the factories. New ideas/elements required for Sprint 3 is a game flow class since GameFlow can be used to combine everyone's functionality implementations.

For Bel's prototype, the main classes were board and chit. Ideas/elements to be used in Sprint 3 is click one chit per turn since flipping just one chit at a time will avoid cheating; it will auto turn back after 3 seconds.

For Ranusha's prototype, the main classes were gameboard and game. Ideas/elements to be used in Sprint 3 is to change game state. New ideas/elements required for Sprint 3 is to refactor win game into function and move it to gameflow from game since the gameflow can check if a player has re-entered their own cave, which is needed to change gamestate to "win" state.

For Ammar's prototype, the main classes were gametoken and player. Ideas/elements to be used in Sprint 3 is to change player turn. New ideas/elements required for Sprint 3 is to move change turn function to gameflow from player since it is more relevant to move this function to gameflow for better readability and maintenance.

In conclusion, Chamille's code will be used as the base as player movement and position tracking covers a large part of the game's implementations. Usage of the factories, Position class and the board's 2D internal array will help with this. Bel, Ammar, and Ranusha will add their implementations from Sprint 2 in the appropriate places.

CRC Cards of six main classes

Player

This class represents each player/token on the board. When the player is initialised, it is given a specific cave where it starts from. Each player also keeps track of consecutive pirate chits clicked. Lastly, it also keeps track if it has finished the game.

Player	
Starting cave	Cave
Check current positiion	Position
Move player to cave at the start	
Track consecutive pirate chits	
Check if player is finished	

AnimalFactory

This class returns all the required instances of animals that are required to be used by the various board pieces. For each type of board piece, it accordingly creates the correct number and type of animals using the specific Animal class (which is the abstract class of the five different types of animals available). For example, for the cave objects, AnimalFactory outputs two, three or four unique animals, depending on the number of players.

AnimalFactory	
Create animal instances required for the chits	Animal
Create animal instances required for the caves	
Create animal instances required for the volcano cards	

BoardFactory

This class returns all the required instances of board piece objects. Each chit, cave and volcano card have corresponding animals, which are taken from AnimalFactory. Then, according to Fiery Dragons specific, each set of board piece objects is created using the specific methods.

BoardFactory	
Create all the instances of the chits Create all the instances of the caves Create all the instances of the volcano cards	AnimalFactory BoardPiece

Board

This class represents the whole board consisting of all the board pieces. It creates all the Player instances and uses BoardFactory to create all the instances of the board pieces. The board is represented using a 2D matrix, each element corresponding to the position of a board piece on the screen. Each piece is given a randomised position, and the display coordinates are calculated based on the piece's position in the matrix. Finally, all board pieces are drawn onto the screen given position in the matrix.

Board	
Chits Volcano cards Caves Players Position and draw all the board pieces (including tokens) onto the screen	BoardFactory Player Position

Main

This is the main game class where pygame is initialised and the instance of the board and game flow are created using the Board and GameFlow classes. The GameFlow class then communicates with the Board class to carry out the game. This class also displays a welcome screen where the user can input the number of players before starting the game.

Main	
Display welcome screen	Board
Number of players	GameFlow
Initialize board	
Run game flow	

GameFlow

This class controls the whole game and implements the template design pattern. Its primary function is to ensure the game loop runs according to the required functionality. It tracks which player is playing currently and listens for clicks on chits, and accordingly moves the player. After the current player's turn is done, the game flow switches the current player. Finally, when a player has won(re-entered their cave after going a round the board), GameFlow displays the 'win game' screen and terminates.

GameFlow	
Current player	Board
Board	Player
Listen for chit clicks	Position
Flips/unflips chits accordingly	
Moves player on the board	
Updates board in game loop	
Change player turn	
End/win game	

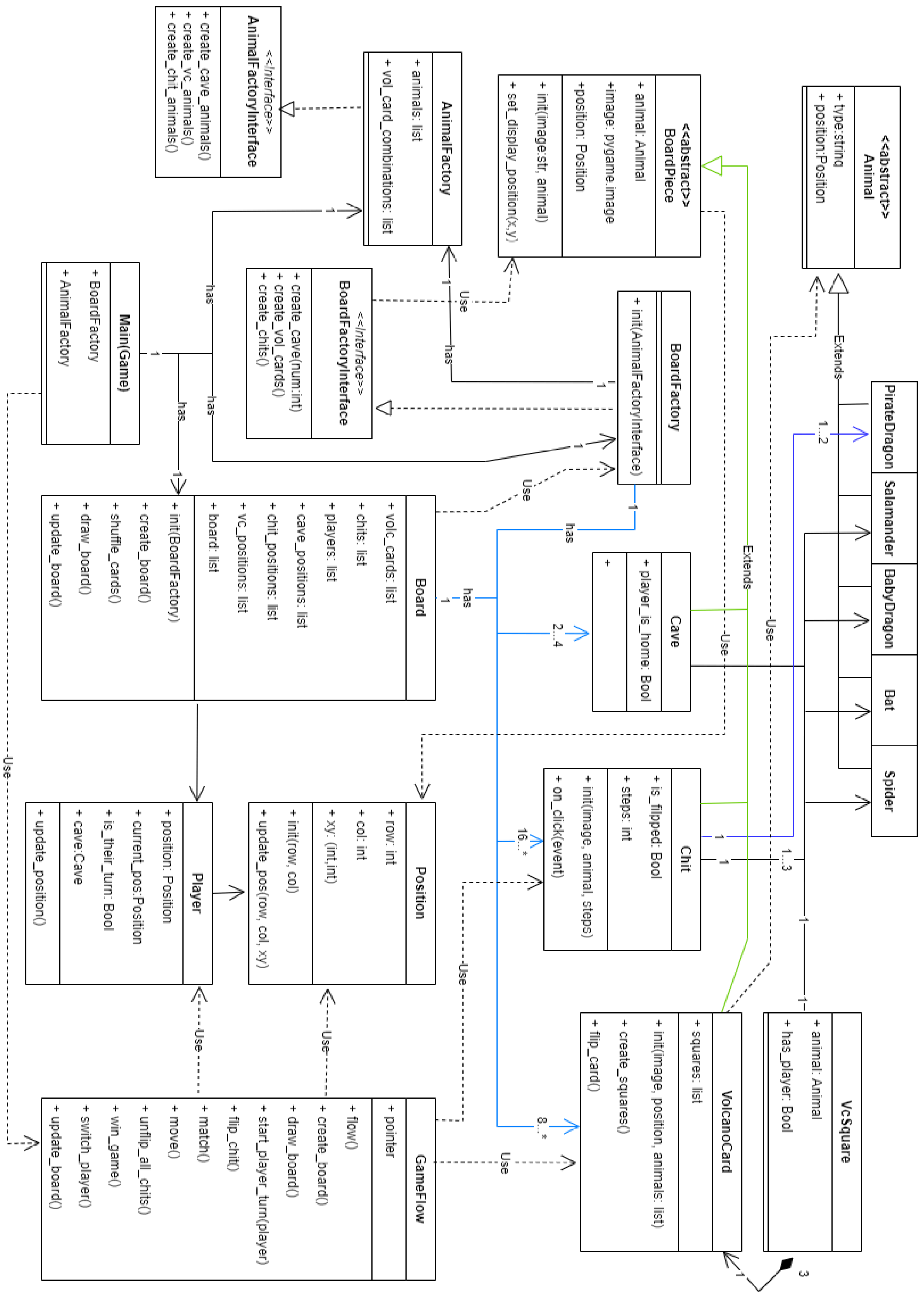
Alternative Distribution of Responsibilities

The team considered combining BoardPieceFactory and AnimalFactory into one class 'Factory', which shares both sets of responsibilities. However, to ensure the code follows the single-responsibility principle, two separate classes were used. Also, BoardPieceFactory depends on AnimalFactory, and having this dependency defined explicitly allows for better understandability and modifiability (suppose we need to add an extra animal for Sprint 4).

Currently, the players are initialised in the Board class. An alternative considered was initialising them in GameFlow since the logic for player turn and game state is defined here. However, since Board is responsible for drawing itself (including players) onto the display and Player has an association with the positions on the board, it is much easier to implement to create players in Board, hence resulting in encapsulation.

Initially, flipping chits and moving players were implemented in the Board class. However, placing all these methods in the same class makes it over-complex. This causes the code not following many notable software qualities, particularly maintainability and flexibility. Therefore, we decided that all chit flipping functionalities will be implemented in gameflow.

Class diagram of consolidated design



Description of executable

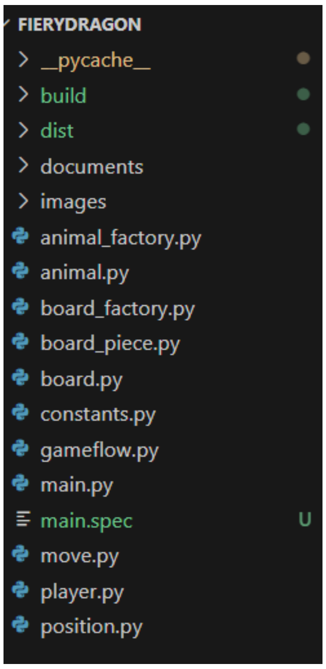
To distribute the Python game application with a Pygame GUI as a standalone executable, we used PyInstaller. Here are the steps and commands involved in creating the executable:

1. Install PyInstaller:
   ```
   pip install pyinstaller
   ```
   We used pip to install the PyInstaller in the Python environment.

2. Code Implementation:
   Our implementation for the application was using a Python script where our main file was ‘main.py’. The script includes the PyGame GUI logic.

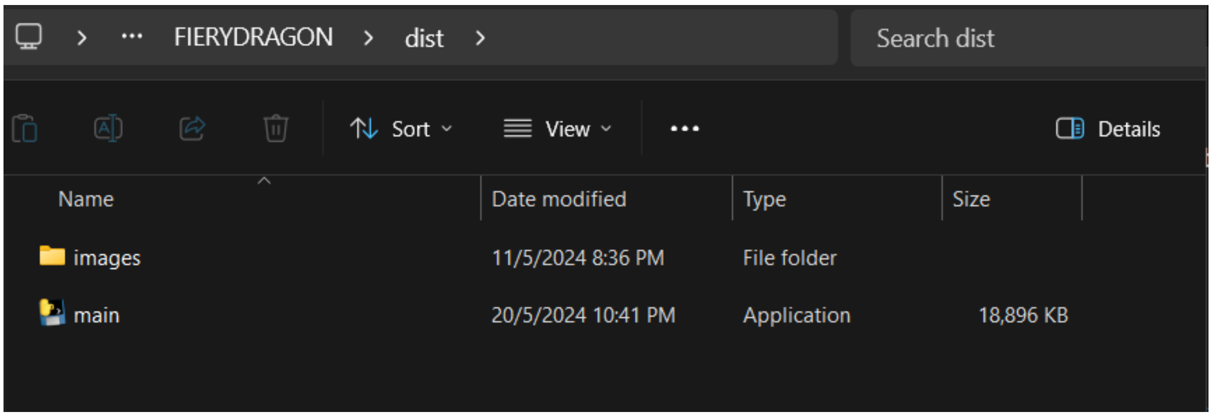
3. Creating the executable:
   We used the PyInstaller to bundle the script into a single executable file by having the ‘--onefile’ option that ensures that all dependencies are packaged into a single executable.
   ```
   pyinstaller main.py --onefile
   ```

4. Output files:
   The PyInstaller will generate several files and directories which consist of:
   - ‘dist/main.exe’ : The standalone executable file
   - ‘build/’ : A directory containing temporary files used during the building process
   - ‘main.spec’: A spec file that describes how to build the executable which is customizable for advanced configurations



```
FIERYDRAGON
> __pycache__
> build
> dist
> documents
> images
animal_factory.py
animal.py
board_factory.py
board_piece.py
board.py
constants.py
gameflow.py
main.py
main.spec        U
move.py
player.py
position.py
```

5. Execution of game:
   We made sure the images folder where all the relevant images are within the 'dist' folder in order for a proper display of the images used within the application.



Name	Date modified	Type	Size
images	11/5/2024 8:36 PM	File folder	
main	20/5/2024 10:41 PM	Application	18,896 KB