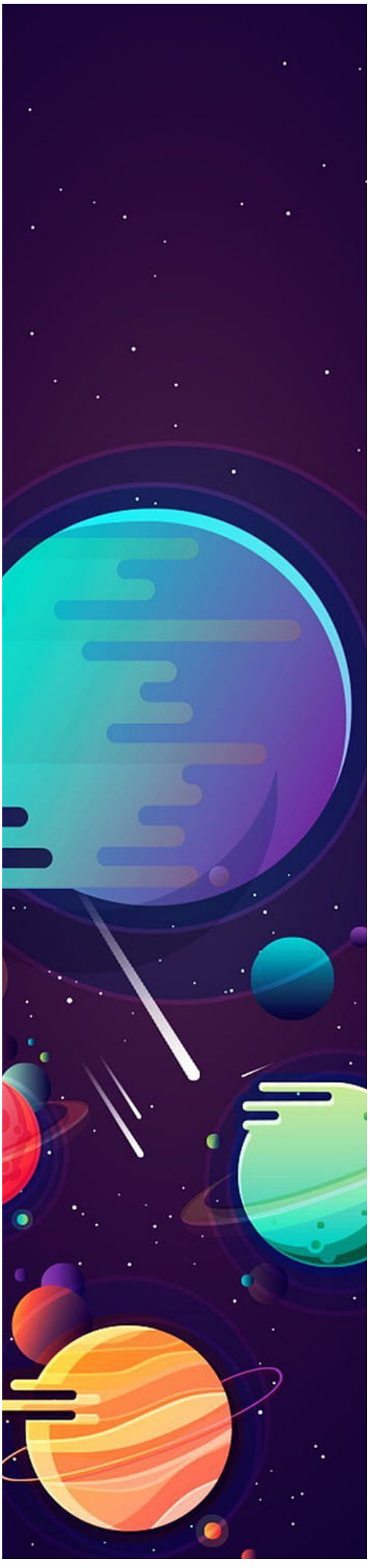


FIT3077
Sprint 4
Design Rationale

MONASH University

Table of content

Revised User Stories

All in green are modified or newly added user stories

As a general player, I was a launch screen that shows a playing option so that I can play the game when I am ready to

As a novice player, I want to have the option to view a tutorial so that I can understand how to play the game before beginning

As a novice player, I want to be able to show hints so that I may know what move to play next if I choose to have help during the game

As an experienced player, I want to be able to turn the hints on and off so that I can choose to play in either state.

As a beginner player, I want a tutorial that shows how to place down tokens so that I know how to begin playing the game

As a beginner player, I want a tutorial that shows how to move tokens so that I know how the moving mechanism works in-game

As a beginner player, I want a tutorial that shows how to form a mill so that I know how to form one during the game

As a beginner player, I want a tutorial that shows how to remove tokens so that I know how to continue the game by removing them

As a beginner player, I want the tutorial to show flying tokens when there are less than three tokens so that I know what the game is supposed to look like when the flying is enabled

As a beginner player, I want the tutorial to show winning by restricting my opponent's moves so that I can see what a possible losing state looks like

As a beginner player, I want the tutorial to show winning by getting my opponent to 2 tokens so that I can understand how to win the game

As a general player, I want to be able to change the game state in the GUI so that I can toggle back between tutorial states

Architecture and Design Rationales

Design Rationales

TutorialState: Should the tutorial be an abstract class or a concrete class

Issue:
Should tutorials be initialised through an abstract tutorialState class or have one concrete tutorial class

Alternative A:
A tutorialState abstract class that contains the relevant abstract methods

Advantages:

- Allows for extensibility of code as new tutorials can be added easily by inheriting the abstract tutorialState class and new methods for specific tutorials can be added
- Game depends on abstraction as opposed to concrete tutorial class, which adheres to SOLID principles
- Provides superior flexibility for each tutorial, as each tutorial can implement their own specific conditions

Disadvantages:

- Harder to implement, as there are multiple abstract methods that must be implemented in order to have a working tutorial

Alternative B:
Have a concrete tutorial class

Advantages:

- Easier to implement as each tutorial only has to input specific parameters for the constructor, as opposed to implementing the required abstract methods

Disadvantages:

- Provides no extensibility and flexibility, as any change to one tutorial will be applied and refactored to every other tutorial
- Game depends on concretions, which makes handling different and new tutorials more difficult, and does not adhere to good SOLID practices.
- The tutorial class will end up taking on too much responsibility, as it will handle specific conditions for every tutorial.

Decision:
The clear winner for this decision was Alternative A, which was to create a tutorialState abstract class instead of a concrete tutorial class to handle tutorials. Although implementing tutorials through a concrete tutorial class would have been much easier to implement, the benefits from abstraction, such as better extensibility, flexibility, and adhering to SOLID principles, made Alternative A the superior option.

TutorialManager: Should there be a separate class to manage tutorials?

Issue:
Should there be a new class tutorialManager be used to manage individual tutorials, or should game manage tutorials?

Alternative A:
A tutorialManager class is created to manager tutorials

Advantages:
- Allows for specific methods and behaviours to be implemented to handle tutorial management
- Adheres to single responsibility principle, as tutorial management responsibility is delegated to new class as opposed to being managed by the Game

Disadvantages:
- Harder to implement as tutorialManager will have to set specific conditions for Game and GameBoardGui to manage tutorials effectively,

Alternative B:
Have Game class manage tutorial

Advantages:
- Easier to implement as the game has direct access to all tutorials.

Disadvantages:
- Violates single responsibility principle as Game class is taking on too much responsibility, and should not manage tutorials as well.
- Tutorials cannot be accessed outside of Game class, meaning unnecessary dependencies will form to access tutorials.

Decision:
Although Alternative B is easier to implement, Alternative A was chosen. Alternative implementing a tutorialManager class was selected as it allows us to implement specific methods and behaviours that might be needed to manage tutorials. This option also adheres to the single responsibility principle, as tutorial management is delegated to tutorialManager as opposed to Game managing everything. Thus, alternative A chosen to handle tutorial management.

TutorialMode: Should there be a separate class for tutorial mode gameplay?

Issue:
Should the tutorial mode be its own separate class, or should game implement capabilities to handle tutorial mode

Alternative A:
Game has tutorial mode capability

Advantages:

- Using capabilities allows for easy extension for future game modes, as new game mode enums can be made to represent new game state.
- Prevents the reuse of code, as gameplay rules do not change in tutorial mode. Furthermore, any updates made in game will automatically transfer to tutorial game play

Disadvantages:

- Harder to implement as game state has to be applied where appropriate to initiate normal vs tutorial game modes.

Alternative B:
Separate tutorial class

Advantages:

- Easier to implement as a new tutorial game mode can be made from scratch
- Provides some flexibility in adding new rules for gameplay

Disadvantages:

- Code will be repeated as tutorial game play and normal game play rules are identical
- Refactoring game will require refactoring tutorial in order to maintain consistency, which can introduce bugs
- Limited future extensibility as if a new game mode is to be made, a whole new class will have to be constructed from scratch

Decision:
Although alternative B provides additional flexibility in adding new rules for gameplay, ultimately, alternative A provided a better solution for our implementation. Using capabilities allows for easy extension in adding new game modes in the future, and also prevents the reuse of code, as the game play rules are identical in tutorial mode and in normal game mode. Thus alternative A was the more appropriate choice for adding a tutorial mode.

Token Cover on the Game Board GUI

Issue:
The player needs an indication of how many tokens are left in the token bank so they can place tokens at the start of the game

Alternative A:
Make the tokens each a label and get rid of the image path each iteration

Advantages:
- A quicker implementation as the button class is already predesigned and just needs to be modified

Disadvantages:
- Have to manually align each of the 18 tokens to its position and line them up
- Makes the code slower to function as each iteration the token needs to be accessed and the image path needs to be updated
- Will have to modify the currently existing button function to implement the feature accurately
- Needs to access 18 instances of the image to make it appear on the button image path

Alternative B:
Have one label with an image over it and each iteration the image path gets updated with a cover to hide the used tokens

Advantages:
- Only need 9 cover images of each stage of the token place process
- Need only two labels that need to be updated each iteration
- Makes the code less complex and in turn faster
- Need to only align the labels to the token image once
- Doesn't modify the current code and only extends the same code

Disadvantages:
- A slower implementation process that needs trial and error for positioning

Decision:
Alternative B was chosen as the approach to covering the used tokens on the game board. This was chosen as it adhered to the SOLID principles, particularly the open/closed principle as the other approach modifies the current code which could cause bugs while Alternative B looks to extend the current code, making it easier to debug. Though it is a slower implementation process, the cons of alternative A violate too many object-oriented practices and make the game very slow when implemented due to the various time it but add and remove buttons.

Tutorial State Finalisation

Issue:
Should the tutorial implementation be a full game where the user is restricted to the

moves the system will let them or be able to play each move of the game at their own pace?

Alternative A:
Make the game board static and have only a few certain tokens that can move and the player can toggle between game states to see how each move is played

Advantages:
- A simple implementation of the Swing GUI recalling a different image for the gameboard each time the button is toggled
- This implementation allows for only one JPanel for the tutorial game instead of slowing the system down with multiple over and over again
- Only a certain number of tokens will be placed on the board so the complexity is low and the game will run faster
- The user is given free will to go through each game state they want to look as instead of replaying the whole game each time to see how a certain rule is played

Disadvantages:
- The game isnt as dynamic as the team initially hoped it would be
- Strays away from the original implementation of the advanced feature the team had discussed during Sprint 1

Alternative B:
Make the user play a whole restricted game where the player will move the tokens the system will allow for despite all pieces of the board being available to select

Advantages:
- Dynamic game play that the user can play and will transition into the actual nine mens morris game
- Will be using the code that is already implemented to create the game board and add restrictions

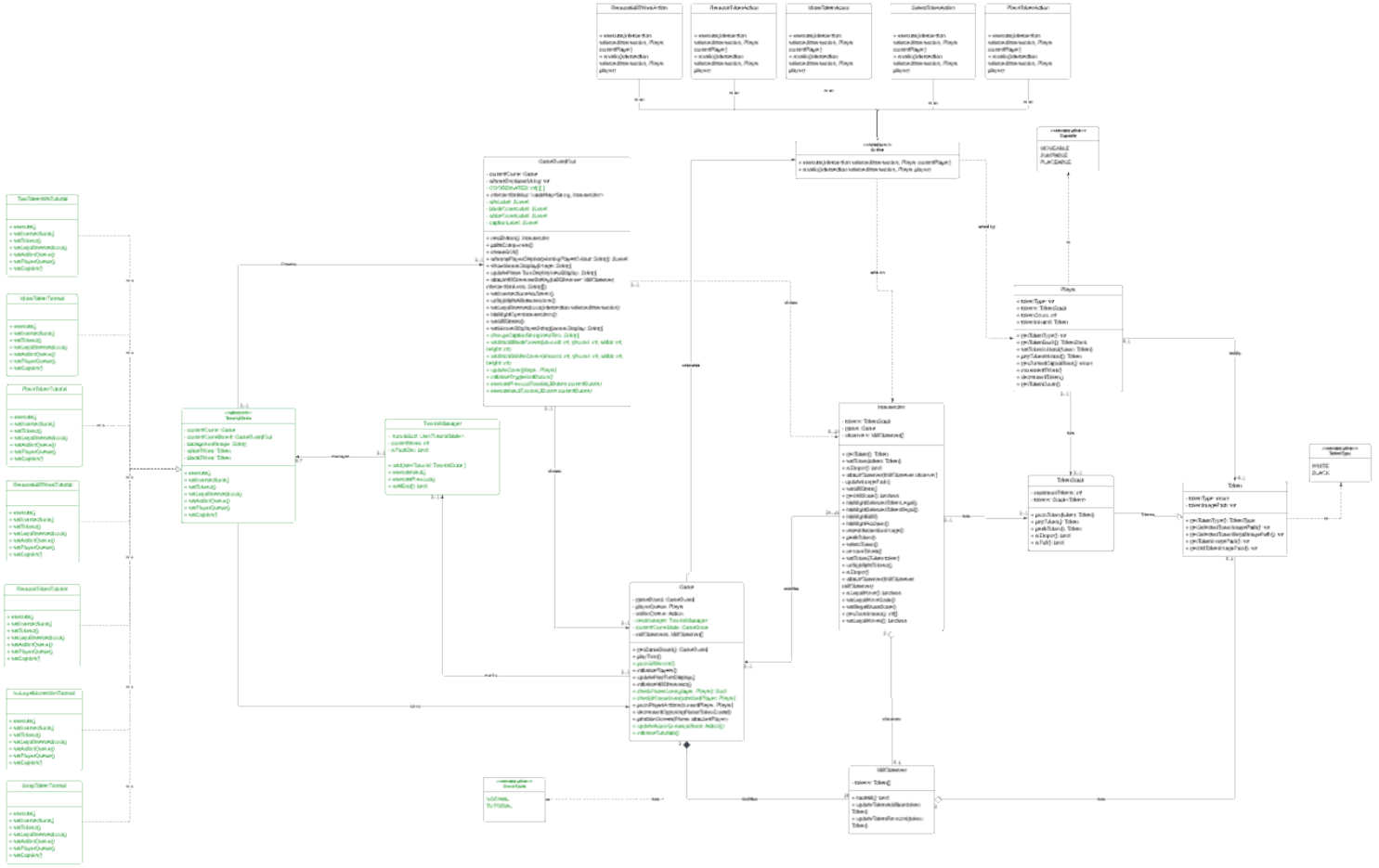
Disadvantages:
- A slower implementation process with a lot classes needing to be modified which violates open/closed principle
- The game will be much slower as the whole gameboard is running again during the tutorial and then the game must lift off restrictions when the game actually begins
- Users have no free will during the tutorial mode and must play the game as the system provides.

Decision:
Our final implementation of the tutorial mode advanced feature was completely different to the initial discussion during sprint 1 yet it showcased a lot of good design practices and adhered to most solid principles in good object-oriented programming. Alternative A was the implementation the team ended up choosing as this allowed us to create an abstract tutorial state class that each tutorial can extend. This allows the game board for the actual game to stay unmodified adhering to the open and closed principle and only being called to create the game background for each tutorial state. Though this option is

not as dynamic as the team initially thought, it was an easier way to implement it without a lot of bugs as it used code that was already there and extended it instead of writing new code or modifying one that was already there. This also made the game fast and easier to work on Swing with.

Overall the implementation we chose to use for the advanced requirement feature turned out to be less difficult than we had initially planned. This was mostly because the code we used was already implemented and used the same idea as the Action abstract class and the action queue which managed each type of action. This means the code was reusable and we didn't need to modify a lot of code to integrate it. This implementation also helped the team to adhere to SOLID principles as we aligned with the Do Not Repeat Yourself principle by using the abstract class and the Liskov Substitution Principle as the TutorialState classes that extended the abstract class can replace one another in the code. This made implementation very simple as we only needed one code to handle all the possible tutorial states. This also appeals to open/closed principle as the tutorial manager class made it so that new tutorials can be added in the future without modifying current code and simply just adding it as an extension of the tutorial manager.

Revised Class Diagram



Appendix A: Design Rationale Guidelines

Guidelines

Guidelines for Writing Good Design Rationales

- What is the Decision Point (what are you trying to decide)?
- What are the design alternatives/options?
- What are the advantages and disadvantages of each?
- What is the final decision based on above considerations?

An Example

Issue: A single player and a multiplayer are very similar classes. Should we create an abstract class, Player, that encompasses the common features?

Alternative A: Create a Player abstract class.

Advantages:

- Promotes reuse
- Makes the solution easy to extend should there be some other type of gameboard in the future.
- May contribute to simpler algorithms via polymorphism.

Disadvantages: Adds a layer of abstraction to the design.

Alternative B: No Player abstract class.

Advantages: Simpler design.

Disadvantages:

- May not promote the right reuse
- May have to write similar code for common features in two places.
- More work to extend in the future to accept different board type.

Decision: Alternative A. The advantages of A outweigh the disadvantages, while Alternative B has more disadvantages than advantages.

Template

Issue:

Alternative A:

Alternative B:

Advantages: **Disadvantages:**	**Advantages:** **Disadvantages:**

Decision: