

FIT 3077 Software Engineering: Architecture and Design

Sprint Three Documentation

Group Name: Stack Underflow

(MA_Wednesday04pm_Team128)

Members :

Chang Yi Zhong (33991499)

Nikhita Peswani (31361552)

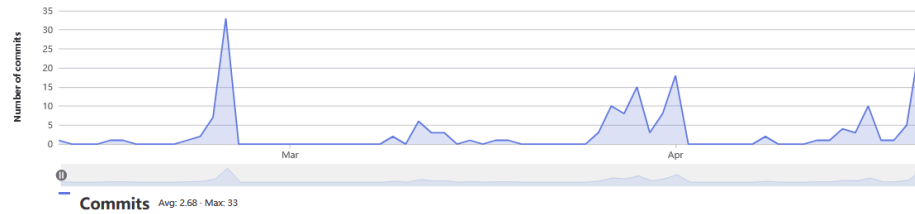
Lim Hung Xuan (33984972)

Enrico Tanvy (33641668)

Contribution analytics:

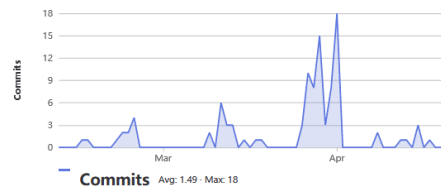
Commits to ycha_v2_branch

Excluding merge commits. Limited to 6,000 commits.



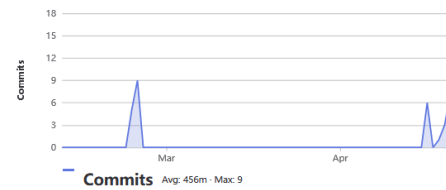
Chang Yi Zhong

101 commits (ycha0154@student.monash.edu)



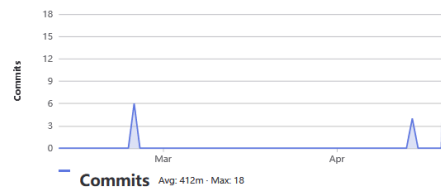
hlim0069

31 commits (hlim0069@student.monash.edu)



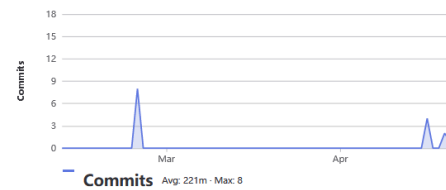
Nikhita Peswani

28 commits (npes0001@student.monash.edu)



enri0002

15 commits (enri0002@student.monash.edu)



1. Review of Sprint 2 Tech-based Software Prototypes

1.1 Defining Assessment Criteria and the corresponding metrics

Review based on the assessment criteria:

Factor	Characteristic	Design Principle	Metric	Achieved?
Functional Suitability	Functional completeness	set up the initial game board (including randomized positioning for dragon cards)	the volcano cards that make up the gameboard are shown	
			the chit cards are shown	
			the caves are shown	
		flipping of dragon cards	the chit card shows the hidden image after being clicked	

		movement of dragon tokens based on their current position as well as the last flipped dragon card	the token is on the correct tile after a chit card is flipped	
		change of turn to the next player	the turn is changed to the next player	
		winning the game	the winner is determined	
			game is ended	
		the game accommodates 2 to 4 players	the game can be played with 2 to 4 players	
			the corresponding number of caves are shown	
	Functional correctness	randomization of the volcano cards	on every run of the game the volcano cards are randomized	
			on every run of the game the original shape of the board is maintained	

			on every run the number of volcano cards is always 8	
		randomization of chit cards	on every run of the game chit cards are randomized	
		chit cards are flipped correctly	chit cards remain flipped until an unmatching chit card is chosen	
			chit cards will be flipped back to their original face-down position once each player's turn ends	
		caves are placed correctly	all caves are equally spaced apart	
			each cave is connected to a volcano tile	
		token are placed correctly	each token starts out in their correct respective cave	

		tokens are moved correctly	the token moves forward if the player flips a matching chit card	
			the number of tiles the token moves matches with the number of animals on the chit card	
			the token stays put if another token is already blocking the tile it is supposed to move to	
			the token moves backwards if the flipped chit card has pirate dragon on it	
			the token goes back to its cave when the player flips a chit card that allows it to move the exact number of tiles to reach its cave, after completing 1 rotation around the board	

	Functional appropriateness	board is displayed in a way that the players can understand	the entire board is displayed to the players without any part of it missing or out of view	
			the chit cards are flipped face-down and are clearly visible in the middle of the board	
			all caves are connected to a volcano tile, evenly spaced and clearly visible on the board	
			all tokens are clearly visible, starting in their respective caves	
		flipping of chit cards according to the game rules	when the player clicks on the chit card, it flips the chit card and uncovers the animal behind it	
			When a player flips an unmatching chit card, the	

			chit card shows the animal for a short while before flipping back	
		Current player's turn is visible on the board	there is visible text that indicates which player's turn it is currently	
		token movement is visible on the game board	the token's movement between the tiles is clearly visible	
		The winner is shown to all players at the end of the game	clear indication of the winner	
		Players gets to choose how many players are going to play before the game starts	the user can increase or decrease the player count before the start of the game using 'plus' and 'minus' buttons	
			the correct number of caves that correspond to the number of players are fully shown and visible on the board	

Interaction Capability	Appropriateness recognizability	the board represents a cyclic game board	the board is cyclical, and connected on both ends	
		each tile of the volcano represents 1 step	uniformed tile sizing	
	User engagement	Player is able to interact with the UI	The user is able to click on the chit cards	
		Player is able to identify that the UI elements are interactable	interactable UI elements are clearly shown and appeals to the user	
Maintainability	Modifiability	Minimal changing of modules to cater modification	Number of modules affected by typical changes $\leq \frac{1}{3}$ of classes; major changes $\leq \frac{1}{2}$ of classes	
		usage of abstract classes and interfaces	common methods are abstracted or in interfaces	
	Modularity	Coupling	Fan-in ≤ 3	
			Fan-out ≤ 5	
			Coupling between object classes (CBO) ≤ 7.5	

	Testability	Reliance on case analysis and /or down-casts	number of downcasts needed in each method ≤ 2 (exception to primitive -> primitive cast provided by java)	
Reliability	Faultlessness	classes is not overly complex	Weighted Methods per Class (WMC) ≤ 6	

Metrics description and justification :

While we primarily rely on our own criteria to assess quality, certain standard metrics are also used to measure various aspects of the software system's quality. Following are the description and also justification of the standard metric that is utilized :

1) Coupling between object (CBO)

CBO metric quantifies the dependencies of a class by assessing factors such as the return types of methods, variable types, and attributes declared within the class. This metric is employed to evaluate the modularity of a system, as a higher CBO value indicates a greater degree of interdependence among classes, suggesting a need for improved modularity.

2) Fan-in and Fan-out

Fan-in and fan-out metrics evaluate the relationships between modules within a software system. Fan-in quantifies the number of modules that directly call a particular module, while fan-out measures the number of modules that a module directly calls. These metrics provide insights into the level of coupling and cohesion within the system. High fan-in suggests that a module is widely reused, indicating good reusability and potentially contributing to better modularity. Conversely, high fan-out may indicate a module with numerous dependencies, potentially reducing modularity due to increased coupling. Therefore, fan-in and fan-out metrics are employed to assess the modularity of a system, with high values suggesting areas for improvement in terms of encapsulation and separation of concerns.

3) Weighted method per classes (WMC)

The Weighted Methods per Class (WMC) metric assesses the complexity of a class by evaluating the complexity of its methods. The calculation of method complexity takes into account the number of decision points, which is marked by if-else statements, for loops, or while loops within the method. This metric is utilized to assess the Faultlessness of the system, as a higher WMC may indicate a greater likelihood of errors occurring at runtime.

Metric scores:

Chang Yi Zhong

WMC average value from all classes: 5.8

Fan-in average value from all classes : 2.5

Fan-out average value from all classes: 4.85

CBO average value from all classes : 7.35

Nikhita

WMC average value from all classes: 7.5

Fan-in average value from all classes : 2.45

Fan-out average value from all classes: 4.81

CBO average value from all classes : 7.27

Hung Xuan

WMC average value from all classes: 5.35

Fan-in average value from all classes : 2.45

Fan-out average value from all classes: 4.1

CBO average value from all classes : 6.55

Enrico

WMC average value from all classes: 4.89

Fan-in average value from all classes : 3

Fan-out average value from all classes: 4.9

CBO average value from all classes : 7.89

1.2 Review of each of the Sprint 2 tech-based software prototypes based on the assessment criteria

1.2.1 Review of the Sprint 2 tech-based software prototypes : Chang Yi Zhong

Review based on the assessment criteria:

Factor	Characteristic	Design Principle	Metric	Achieved?
Functional Suitability	Functional completeness	set up the initial game board (including randomized positioning for dragon cards)	the volcano cards that make up the gameboard are shown	✓
			the chit cards are shown	✓
			the caves are shown	✓
		flipping of dragon cards	the chit card shows the hidden image after being clicked	✓

		movement of dragon tokens based on their current position as well as the last flipped dragon card	the token is on the correct tile after a chit card is flipped	✓
		change of turn to the next player	the turn is changed to the next player	✓
		winning the game	the winner is determined	✓
			game is ended	✓
		the game accommodates 2 to 4 players	the game can be played with 2 to 4 players	✓
			the corresponding number of caves are shown	✓
	Functional correctness	randomization of the volcano cards	on every run of the game the volcano cards are randomized	✓
			on every run of the game the original shape of the board is maintained	✓

			on every run the number of volcano cards is always 8	✓
		randomization of chit cards	on every run of the game chit cards are randomized	✓
		chit cards are flipped correctly	chit cards remain flipped until an unmatching chit card is chosen	✓
			chit cards will be flipped back to their original face-down position once each player's turn ends	✓
		caves are placed correctly	all caves are equally spaced apart	✓
			each cave is connected to a volcano tile	✓
		token are placed correctly	each token starts out in their correct respective cave	✓

		tokens are moved correctly	the token moves forward if the player flips a matching chit card	✓
			the number of tiles the token moves matches with the number of animals on the chit card	✓
			the token stays put if another token is already blocking the tile it is supposed to move to	✓
			the token moves backwards if the flipped chit card has pirate dragon on it	✓
			the token goes back to its cave when the player flips a chit card that allows it to move the exact number of tiles to reach its cave, after completing 1 rotation around the board	✓

	Functional appropriateness	board is displayed in a way that the players can understand	the entire board is displayed to the players without any part of it missing or out of view	✓
			the chit cards are flipped face-down and are clearly visible in the middle of the board	✓
			all caves are connected to a volcano tile, evenly spaced and clearly visible on the board	✓
			all tokens are clearly visible, starting in their respective caves	✓
		flipping of chit cards according to the game rules	when the player clicks on the chit card, it flips the chit card and uncovers the animal behind it	✓
			When a player flips an unmatching chit card, the	✓

			chit card shows the animal for a short while before flipping back	
		Current player's turn is visible on the board	there is visible text that indicates which player's turn it is currently	✓
		token movement is visible on the game board	the token's movement between the tiles is clearly visible	✓
		The winner is shown to all players at the end of the game	clear indication of the winner	✓
		Players gets to choose how many players are going to play before the game starts	the user can increase or decrease the player count before the start of the game using 'plus' and 'minus' buttons	
			the correct number of caves that correspond to the number of players are fully shown and visible on the board	✓

Interaction Capability	Appropriateness recognizability	the board represents a cyclic game board	the board is cyclical, and connected on both ends	✓
		each tile of the volcano represents 1 step	uniformed tile sizing	✓
	User engagement	Player is able to interact with the UI	The user is able to click on the chit cards	✓
		Player is able to identify that the UI elements are interactable	interactable UI elements are clearly shown and appeals to the user	✓
Maintainability	Modifiability	Minimal changing of modules to cater modification	Number of modules affected by typical changes $\leq \frac{1}{3}$ of classes; major changes $\leq \frac{1}{2}$ of classes	✓
		usage of abstract classes and interfaces	common methods are abstracted or in interfaces	✓
	Modularity	Coupling	Fan-in ≤ 3	✓
			Fan-out ≤ 5	✓
			Coupling between object classes (CBO) ≤ 7.5	✓

	Testability	Reliance on case analysis and /or down-casts	number of downcasts needed in each method ≤ 2 (exception to primitive \rightarrow primitive cast provided by java)	✓
Reliability	Faultlessness	classes is not overly complex	Weighted Methods per Class (WMC) ≤ 6	✓

Summary of the key finding:

Functional Sustainability

In terms of functional sustainability this particular prototype ticks all the discussed metrics where the key functionality for Fiery Dragon according to the basic game rule is fully achieved, with the only thing lacking being the ability for the user to increase or decrease the number of players to a minimum of 2 or a maximum of 4 with a user interface.

Interaction Capability

As for the interaction capability of this prototype, it is highly recognizable to the original game and it is clear on it being a board game where the board game is similar in shape and arrangement to the original Fiery Dragon and the volcano cards are of uniformed size. In terms of its user engagement, as per of metrics it is highly engageable with its appealing colors and responsive ui elements.

Maintainability

Looking at the maintainability, we deemed it highly modifiable with the usage of abstract classes and interfaces when common services are abundant and when minor changes such as attribute change and logic changes without affecting return types or major changes that will alter the return type or outright change a class internal logic, affected classes are within group agreed limits. As for modularity and testability, we rely on a metric that requires the calculation of the prototype code and for this prototype, has relatively low values for the functionality it provides.

Reliability

The reliability of the prototype is deemed highly reliable following our metric that utilizes the weighted method per class number where it is the sum of complexity of methods in a class. With the WMC threshold it achieved, it is functional enough for the system in question, while maintaining a low overall complexity to reduce faults appearing in the design.

1.2.2 review of the Sprint 2 tech-based software prototypes: Nikhita

Review based on the assessment criteria:

Factor	Characteristic	Design Principle	Metric	Achieved?
Functional Suitability	Functional completeness	set up the initial game board (including randomized positioning for dragon cards)	the volcano cards that make up the gameboard are shown	✓
			the chit cards are shown	✓
			the caves are shown	✓
		flipping of dragon cards	the chit card shows the hidden image after being clicked	✓
		movement of dragon tokens based on their current position as well as the last flipped dragon card	the token is on the correct tile after a chit card is flipped	✓
		change of turn to the next player	the turn is changed to the next player	✓
		winning the game	the winner is determined	✓
			game is ended	

		the game accommodates 2 to 4 players	the game can be played with 2 to 4 players	
			the corresponding number of caves are shown	✓
	Functional correctness	randomization of the volcano cards	on every run of the game the volcano cards are randomized	✓
			on every run of the game the original shape of the board is maintained	✓
			on every run the number of volcano cards is always 8	✓
		randomization of chit cards	on every run of the game chit cards are randomized	✓
		chit cards are flipped correctly	chit cards remain flipped until an unmatching chit card is chosen	

			chit cards will be flipped back to their original face-down position once each player's turn ends	✓
		caves are placed correctly	all caves are equally spaced apart	✓
			each cave is connected to a volcano tile	✓
		token are placed correctly	each token starts out in their correct respective cave	✓
		tokens are moved correctly	the token moves forward if the player flips a matching chit card	✓
			the number of tiles the token moves matches with the number of animals on the chit card	✓
			the token stays put if another token is already blocking the tile it is supposed to move to	

			the token moves backwards if the flipped chit card has pirate dragon on it	✓
			the token goes back to its cave when the player flips a chit card that allows it to move the exact number of tiles to reach its cave, after completing 1 rotation around the board	✓
	Functional appropriateness	board is displayed in a way that the players can understand	the entire board is displayed to the players without any part of it missing or out of view	✓
			the chit cards are flipped face-down and are clearly visible in the middle of the board	✓
			all caves are connected to a volcano tile, evenly spaced and clearly visible on the board	✓

			all tokens are clearly visible, starting in their respective caves	✓
		flipping of chit cards according to the game rules	when the player clicks on the chit card, it flips the chit card and uncovers the animal behind it	✓
			When a player flips an unmatching chit card, the chit card shows the animal for a short while before flipping back	✓
		Current player's turn is visible on the board	there is visible text that indicates which player's turn it is currently	
		token movement is visible on the game board	the token's movement between the tiles is clearly visible	✓
		The winner is shown to all players at the end of the game	clear indication of the winner	

		Players gets to choose how many players are going to play before the game starts	the user can increase or decrease the player count before the start of the game using 'plus' and 'minus' buttons	
			the correct number of caves that correspond to the number of players are fully shown and visible on the board	✓
	Appropriateness recognizability	the board represents a cyclic game board	the board is cyclical, and connected on both ends	✓
		each tile of the volcano represents 1 step	uniformed tile sizing	✓
Maintainability	Modifiability	Minimal changing of modules to cater modification	Number of modules affected by typical changes $\leq \frac{1}{3}$ of classes; major changes $\leq \frac{1}{2}$ of classes	✓
		usage of abstract classes and interfaces	common methods are abstracted or in interfaces	
	Modularity	Coupling	Fan-in ≤ 3	✓
			Fan-out ≤ 5	✓

			Coupling between object classes (CBO) ≤ 7.5	✓
	Testability	Reliance on case analysis and /or down-casts	number of downcasts needed in each method ≤ 2 (exception to primitive -> primitive cast provided by java)	✓
Interaction Capability	User engagement	Player is able to interact with the UI	The user is able to click on the chit cards	✓
		Player is able to identify that the UI elements are interactable	interactable UI elements are clearly shown and appeals to the user	✓
Reliability	Faultlessness	classes is not overly complex	Weighted Methods per Class (WMC) ≤ 6	

Summary of the key finding:

Functional Suitability

The software prototype largely fulfills the functional completeness requirements, except for the game end and gameplay for 2 to 4 players. Most functional correctness requirements are met, but issues remain with chit cards staying flipped until an unmatching chit card is chosen, and tokens being blocked if another token occupies the target tile. The prototype is mostly functionally appropriate, but lacks visible text to indicate the current player's turn, the ability to adjust the player count before the game starts using 'plus' and 'minus' buttons, and a clear indication of the winner. The prototype meets all criteria for appropriateness recognizability metrics, demonstrating that the software effectively addresses user needs in an easily identifiable and comprehensible manner.

Maintainability

The software meets the modifiability criteria, except for the abstraction of common methods into interfaces. This shortfall may compromise the system's modifiability, as the absence of abstract methods and interfaces implies a reliance of high-level modules on low-level ones, rather than both depending on abstractions. This increases the coupling between components, making the codebase less adaptable to future modifications. It fulfills all requirements for modularity and meets the criteria for testability.

Interaction Capability

The prototype has successfully met all expected metrics for user engagement, indicating that the user interface is presented in an appealing manner and encourages interaction from users.

Reliability

The software has a Weighted Method per Class (WMC) of 7.5. Since this value falls outside the expected range, the prototype is considered unreliable. A high WMC value suggests increased system complexity, which raises the likelihood of errors and thus reduces the system's reliability.

1.2.3 review of the Sprint 2 tech-based software prototypes: Enrico

Review based on the assessment criteria:

Factor	Characteristic	Design Principle	Metric	Achieved ?
Functional Suitability	Functional completeness	set up the initial game board (including randomized positioning for dragon cards)	the volcano cards that make up the gameboard are shown	✓
			the chit cards are shown	✓

			the caves are shown	✓
		flipping of dragon cards	the chit card shows the hidden image after being clicked	✓
		movement of dragon tokens based on their current position as well as the last flipped dragon card	the token is on the correct tile after a chit card is flipped	✓
		change of turn to the next player	the turn is changed to the next player	
		winning the game	the winner is determined	✓
			game is ended	✓

		the game accommodates 2 to 4 players	the game can be played with 2 to 4 players	
			the corresponding number of caves are shown	✓
	Functional correctness	randomization of the volcano cards	on every run of the game the volcano cards are randomized	
			on every run of the game the original shape of the board is maintained	✓
			on every run the number of volcano cards is always 8	✓
		randomization of chit cards	on every run of the game chit cards are randomized	✓

		chit cards are flipped correctly	chit cards remain flipped until an unmatching chit card is chosen	
			chit cards will be flipped back to their original face-down position once each player's turn ends	
		caves are placed correctly	all caves are equally spaced apart	✓
			each cave is connected to a volcano tile	✓
		token are placed correctly	each token starts out in their correct respective cave	✓
		tokens are moved correctly	the token moves forward if the player	✓

			flips a matching chit card	
			the number of tiles the token moves matches with the number of animals on the chit card	✓
			the token stays put if another token is already blocking the tile it is supposed to move to	
			the token moves backwards if the flipped chit card has pirate dragon on it	✓
			the token goes back to its cave when the player flips a chit card that allows it to move the exact number of tiles to reach its	✓

			cave, after completing 1 rotation around the board	
	Functional appropriateness	board is displayed in a way that the players can understand	the entire board is displayed to the players without any part of it missing or out of view	✓
			the chit cards are flipped face-down and are clearly visible in the middle of the board	✓
			all caves are connected to a volcano tile, evenly spaced and clearly visible on the board	✓
			all tokens are clearly visible, starting in their respective caves	✓

		flipping of chit cards according to the game rules	when the player clicks on the chit card, it flips the chit card and uncovers the animal behind it	✓
			When a player flips an unmatching chit card, the chit card shows the animal for a short while before flipping back	
		Current player's turn is visible on the board	there is visible text that indicates which player's turn it is currently	
		token movement is visible on the game board	the token's movement between the tiles is clearly visible	✓
		The winner is shown to all players at the end of the game	clear indication of the winner	✓

		Players gets to choose how many players are going to play before the game starts	the user can increase or decrease the player count before the start of the game using 'plus' and 'minus' buttons	
			the correct number of caves that correspond to the number of players are fully shown and visible on the board	✓
Interaction Capability	Appropriateness recognizability	the board represents a cyclic game board	the board is cyclical, and connected on both ends	✓
		each tile of the volcano represents 1 step	uniformed tile sizing	✓
	User engagement	Player is able to interact with the UI	The user is able to click on the chit cards	✓

		Player is able to identify that the UI elements are interactable	interactable UI elements are clearly shown and appeals to the user	✓
Maintainability	Modifiability	Minimal changing of modules to cater modification	Number of modules affected by typical changes $\leq \frac{1}{3}$ of classes; major changes $\leq \frac{1}{2}$ of classes	✓
		usage of abstract classes and interfaces	common methods are abstracted or in interfaces	
	Modularity	Coupling	Fan-in ≤ 3	✓
			Fan-out ≤ 5	✓
			Coupling between object classes (CBO) ≤ 7.5	
	Testability	Reliance on case analysis and /or down-casts	number of downcasts needed in each method ≤ 2 (exception to	✓

			primitive -> primitive cast provided by java)	
Reliability	Faultlessness	classes is not overly complex	Weighted Methods per Class (WMC) ≤ 6	✓

Summary of the key finding:

Functional completeness

While the prototype successfully incorporates most of the expected functions outlined in the requirements, it falls short in effectively managing player turns and accommodating a sufficient number of players. Specifically, it struggles to transition between player turns and is limited to supporting only four players, failing to fulfill the metric of "the turn is changed to the next player" and also "the game can be played with 2 to 4 players"

Functional correctness

One primary functionality that the prototype fails to execute correctly is the logic governing the flipping of chit cards. Ideally, chit cards should remain flipped until an unmatched chit card is chosen, and revert to their original face-down position at the end of each player's turn. However, the prototype does not perform this function correctly.

Functional appropriateness

The prototype fails to meet several metrics of functional appropriateness. Firstly, it neglects to briefly display the animal associated with an unmatched chit card. Additionally, there is a lack of textual indication to clarify whose turn it currently is during gameplay. Lastly, the absence of buttons to adjust the player count before initiating the game.

Appropriateness recognizability

The prototype meets all the criteria for appropriateness recognizability metrics, indicating that the software effectively caters to user needs in a manner that is easily identifiable and comprehensible

Modifiability

For modifiability, while the number of modules impacted by changes falls within the specified metric, the software prototype exhibits limited reliance on abstract methods and interfaces. This lack of the utilization of abstract method and interface may compromise the system's modifiability, as the absence of abstract method and interface utilization implies a reliance of high-level modules on low-level ones, rather than both depending on abstraction. This would increase the coupling between components, which makes the codebase less adaptable to future modifications.

Modularity

The prototype fell short in achieving one of the three expected metrics values used to evaluate code modularity: specifically, the metric assessing object coupling. This suggests a potential deficiency in system modularization, as heightened coupling between modules typically corresponds to decreased modularity.

User engagement

The prototype has successfully met all expected metric evaluation for the quality characteristic of user engagement. This indicates that the system's user interface is presented in an inviting manner, encouraging interaction from users

Reliability

With the value of the weighted method per class falling within the expected range, the prototype is considered reliable. This metric assesses the complexity of the system, and a lower value indicates a reduced likelihood of errors, thereby enhancing the system's reliability.

1.2.4 review of the Sprint 2 tech-based software prototypes: Hung Xuan

Review based on the assessment criteria:

Factor	Characteristic	Design Principle	Metric	Achieved?
Functional Suitability	Functional completeness	set up the initial game board (including randomized positioning for dragon cards)	the volcano cards that make up the gameboard are shown	✓
			the chit cards are shown	✓
			the caves are shown	✓
		flipping of dragon cards	the chit card shows the hidden image after being clicked	✓
		movement of dragon tokens based on their current position as well as the last flipped dragon card	the token is on the correct tile after a chit card is flipped	✓

		change of turn to the next player	the turn is changed to the next player	
		winning the game	the winner is determined	✓
			game is ended	✓
		the game accommodates 2 to 4 players	the game can be played with 2 to 4 players	✓
			the corresponding number of caves are shown	✓
	Functional correctness	randomization of the volcano cards	on every run of the game the volcano cards are randomized	
			on every run of the game the original shape of the board is maintained	✓
			on every run the number of volcano cards is always 8	✓
		randomization of chit cards	on every run of the game chit cards are randomized	✓

		chit cards are flipped correctly	chit cards remain flipped until an unmatching chit card is chosen	✓
			chit cards will be flipped back to their original face-down position once each player's turn ends	✓
		caves are placed correctly	all caves are equally spaced apart	✓
			each cave is connected to a volcano tile	✓
		token are placed correctly	each token starts out in their correct respective cave	✓
		tokens are moved correctly	the token moves forward if the player flips a matching chit card	✓
			the number of tiles the token moves matches with the number of animals on the chit card	✓

			the token stays put if another token is already blocking the tile it is supposed to move to	✓
			the token moves backwards if the flipped chit card has pirate dragon on it	✓
			the token goes back to its cave when the player flips a chit card that allows it to move the exact number of tiles to reach its cave, after completing 1 rotation around the board	✓
	Functional appropriateness	board is displayed in a way that the players can understand	the entire board is displayed to the players without any part of it missing or out of view	✓
			the chit cards are flipped face-down and are clearly	✓

			visible in the middle of the board	
			all caves are connected to a volcano tile, evenly spaced and clearly visible on the board	✓
			all tokens are clearly visible, starting in their respective caves	✓
		flipping of chit cards according to the game rules	when the player clicks on the chit card, it flips the chit card and uncovers the animal behind it	✓
			When a player flips an unmatching chit card, the chit card shows the animal for a short while before flipping back	
		Current player's turn is visible on the board	there is visible text that indicates which player's turn it is currently	

		token movement is visible on the game board	the token's movement between the tiles is clearly visible	
		The winner is shown to all players at the end of the game	clear indication of the winner	✓
		Players gets to choose how many players are going to play before the game starts	the user can increase or decrease the player count before the start of the game using 'plus' and 'minus' buttons	
			the correct number of caves that correspond to the number of players are fully shown and visible on the board	
Interaction Capability	Appropriateness recognizability	the board represents a cyclic game board	the board is cyclical, and connected on both ends	✓
		each tile of the volcano represents 1 step	uniformed tile sizing	✓
	User engagement	Player is able to interact with the UI	The user is able to click on the chit cards	✓

		Player is able to identify that the UI elements are interactable	interactable UI elements are clearly shown and appeals to the user	✓
Maintainability	Modifiability	Minimal changing of modules to cater modification	Number of modules affected by typical changes $\leq \frac{1}{3}$ of classes; major changes $\leq \frac{1}{2}$ of classes	✓
		usage of abstract classes and interfaces	common methods are abstracted or in interfaces	✓
	Modularity	Coupling	Fan-in ≤ 3	✓
			Fan-out ≤ 5	✓
			Coupling between object classes (CBO) ≤ 7.5	✓
	Testability	Reliance on case analysis and /or down-casts	number of downcasts needed in each method ≤ 2 (exception to primitive -> primitive cast provided by java)	✓
Reliability	Faultlessness	classes is not overly complex	Weighted Methods per Class (WMC) ≤ 6	✓

--	--	--	--	--

Summary of the key finding:

Functional completeness

The prototype has fulfilled most of the expected features, except for the ability to accommodate 2 to 4 players. Rather, the prototype always accommodates 4 players only. This means that the prototype is unable to cater to one of the key functionalities of allowing 2 to 4 players to play the game, making this prototype functionally incomplete.

Functional correctness

The prototype is mostly functionally correct, it is, however, missing one thing that makes it functionally incorrect, that being the missing functionality of having the volcano cards be randomized at the start of every game. Rather, the prototype always has the volcano cards in fixed positions for every game, this makes the prototype functionally incorrect with regards to this aspect.

Functional appropriateness

Overall, the prototype lacks some of the functional appropriateness. Firstly, the chit card does not show the animal for a short while before flipping back when the player flips an unmatching chit card. Rather, the chit card's image is immediately changed to the back of the chit card, and the player's turn is ended. This is not ideal as the player would have no time to see the chit card they flipped that resulted in their turn ending, which means the player would not be able to memorize the particular chit card's animal. Secondly, there is no visible text that indicates which player's turn it is. Rather, the game just changes the turn of the players behind the scenes. This is not ideal as the players may lose track of whose turn it is, and may end up playing the wrong turn, resulting in a confusing and unfair game. Thirdly, the token's movement between the tiles is not visible. Rather, the token just appears on the destination tile. This is not ideal as having an animation of the tokens actually moving towards the destination would be more clear and understandable to the players. Lastly, the player does not get to choose how many players are going to play before the game

starts. Rather, the game is fixed at 4 players, this not ideal as the physical game of Fiery Dragons is stated to allow 2 to 4 players to play.

Appropriateness recognizability

The prototype is highly recognizable, resembling the physical Fiery Dragons game. The game board is cyclical, connected on both ends, and each volcano tile represents one step, exactly like the physical Fiery Dragons game. This means that the prototype is appropriately recognizable to the players.

Maintainability

In terms of modifiability, the prototype can be easily modified using abstract classes and interfaces, especially when there are numerous common services. This approach allows for changes, like altering logic without impacting the overall code.

In terms of modularity, we use the fan-in, fan-out, and CBO metrics, which were elaborated on at the start of the document, the prototype has relatively low values. In terms of testability, the reliance of downcasts in the prototype is minimal and close to none.

Interaction Capability

In terms of its user engagement, the UI is highly engageable, with UI elements like the chit cards that are obviously interactable to the user. This means that the players are easily able to recognize and interact with the UI elements.

Reliability

The prototype is reliable based on our metric, which uses a weighted method per class number, the metric is elaborated on at the start of the document. The prototype is functional for the system while maintaining low overall complexity to minimize design faults.

What are the ideas/elements of each of the tech-based prototypes from Sprint 2 we are going to use for Sprint 3?

For sprint 3 we decided to use Yi Zhong's design, below is the justification as to why this decision is made.

Among all the prototypes, Yi Zhong's stands out as the most maintainable. While each prototype generally meets the specified assessment criteria, Yi Zhong's design surpasses the rest in terms of maintainability. Firstly, his design exhibits strong modularity, evident in the appropriate separation of system components. Each component operates fairly independently, facilitating easy adaptation to changes. Modifications can be made to individual modules without causing ripple effects throughout the entire system, highlighting Yi Zhong's thoughtful and effective approach to design. Secondly, in terms of modifiability, Yi Zhong's prototype stands out as exceptionally adaptable when analyzing its design, surpassing mere metrics which may not convey the full picture. Its code can be seamlessly and efficiently modified to accommodate extended requirements. While other prototypes may also handle requirement extensions, Yi Zhong's prototype excels in its ability to effectively and efficiently cater to a broader range of extensions.

On top of his code having excellent maintainability, his functionality also stands out as the most comprehensive and complete out of the group. His design covers the vast majority of the basic requirements, and also accommodates for the most extension features down the line. His overall functional suitability stood out as the best, covering basically all the metrics shown above. This is in contrast to the rest of the designs, which lack varying degrees of key functionalities.

Originally, we tried to consolidate all the designs, however we came to the consensus that consolidating our vastly different designs, each dependent on key classes that the other designs may not even have, would result in a worse overall design than if we were to just use Yi Zhong's design directly. This is due to the fact that adding the key classes that are required in order to incorporate the functionality of a specific group member, would, more often than not lead to us breaking one of the design principles

or degrade the final design's performance in our comparison metrics. This, in addition to the justifications given above is why our team decided to use Yi Zhong's design.

What new ideas/elements do we need to bring in in order to improve the prototype?

To improve the prototype, we decided to implement the StageController as a Singleton. This decision ensures that all scene drawing is managed through a single instance of StageController. Without the Singleton pattern, dependency injection would become increasingly complex as more classes requiring the StageController for drawing are added, leading to potential maintenance challenges.

2. Object Oriented Design

2.1 CRC Cards

GameBoard		GameLevel
Responsibilities	Collaborators	
draw game objects	GameObject<<abstract>>	
sets the token's new position based on the flipped chit card	PlayerController	
executes token movement based on flipped chit cards	PlayerController	
determines the winner	GameModeBase	
knows coordinate of each Cave		
knows coordinate of each Tile		
invokes the next turn	TurnController	

Description of purpose:

The gameboard is the class where graphical assets for the game are instantiated, it handles any board centric logic such as

movement or deciding the win condition as it is the main class with a reference link to gameMode that handles the logic when a player has won, and it is the main class that holds reference to all gameobjects. This class is crucial for maintaining the visual and logical integrity of the game and serves as the main point of interaction with other core components such as GameModeBase, TurnController, and various entities like players and cards.

CardController	
Responsibilities	Collaborators
notifies the Gameboard that a card is clicked updates the UI based on the state of the card	ICardDelegate<<interface>>

Description of purpose:

The CardController class manages the individual cards within the game, handling interactions and animations when a card is clicked. It notifies the GameBoard when a card is clicked by using the ICardDelegate interface, which the GameBoard implements. This class is responsible for displaying the card's front and back images, managing click events, and animating the card flipping process. The CardController class plays a crucial role in managing the visual and interactive aspects of cards within the game, ensuring a smooth and engaging user experience by handling animations and interactions efficiently.

BasicGameMode	
Responsibilities	Collaborators
Initializes the game	GameObject<<abstract>>, TurnController
displays the winner of the game	GameWinUI

Description of purpose:

The BasicGameMode class defines a specific game mode with a provided set up of the game. It manages game initialization and also the conditions of winning the game. It handles the responsibility of displaying the winner of the game when a winner is determined. The BasicGameMode class is crucial for setting up, managing, and concluding the game in a structured and coherent manner. It ensures that the game logic is correctly implemented and that the user interface responds appropriately to different game states.

TurnController	
Responsibilities	Collaborators
knows the current player turn updates the player turn	

Description of purpose:

The TurnController class manages the turn logic for the game. It keeps track of the current player's turn and updates the turn to the next player. It is used by the GameBoard to invoke the next turn. This class ensures that the game progresses in a controlled manner by managing the order in which players take their turns. The TurnController class is essential for managing the game's turn-based logic, ensuring that each player gets their turn in the correct order and that the game progresses smoothly from one turn to the next.

PlayerController	
Responsibilities knows which tile each player's token is on knows which cave each player's token belongs to knows how many tiles each player needs to move in order to win	Collaborators

Description of purpose :

The PlayerController class keeps track of the position of each player's token on the game board, which cave each player's token belongs to, as well as how many tiles each player needs to move to get back to their original cave and win. This class manages the player's state within the game level, ensuring that player movements and game logic are accurately tracked and updated. It ensures that player movements are accurately tracked and that the game logic related to player positions is correctly implemented.

StageController	
Responsibilities	Collaborators
controls which scene is shown to the players	ISceneHolder<<interface>>

Description of purpose:

The StageController class is used to set and control the stage that is displayed to the players. It serves as a singleton, ensuring that only one instance of the class manages the primary stage of the application. This class is responsible for configuring the stage's size, resolution scaling, and managing the currently active scene. For example, it is used by BasicGameMode to show the GameWinUI when a winner is determined. Overall, the StageController class centralizes the management of the primary stage, ensuring consistent handling of the game's graphical output and providing a clear structure for scene transitions and stage configuration.

The CardController, BasicGameMode, TurnController, PlayerController, and StageController classes each have a specific responsibility by design. We isolate the specific information and logic needed for each crucial aspect of the game into these respective classes, thereby shifting the responsibility away from the GameBoard and preventing it from becoming a "God Class." By ensuring that each class has a single, focused responsibility, we adhere to the Single Responsibility Principle (SRP). This design ensures that each class handles its responsibilities independently, leading to a well-organized and maintainable codebase where the distribution of responsibilities is clear and appropriate. By structuring the game in this way, each class can be developed, tested, and maintained independently, ensuring a more robust and flexible game architecture.

Alternative distribution:

TokenController	
Responsibilities	Collaborators
knows which tile each player's token is on knows which cave each player's token belongs to knows how many tiles each player needs to move in order to win knows the current player turn updates the player turn	

In an alternative design that we considered, the TurnController and PlayerController are not separated into two distinct classes. Rather, the responsibilities of player's turn management and player's token information are combined and handled by a single class called TokenController.

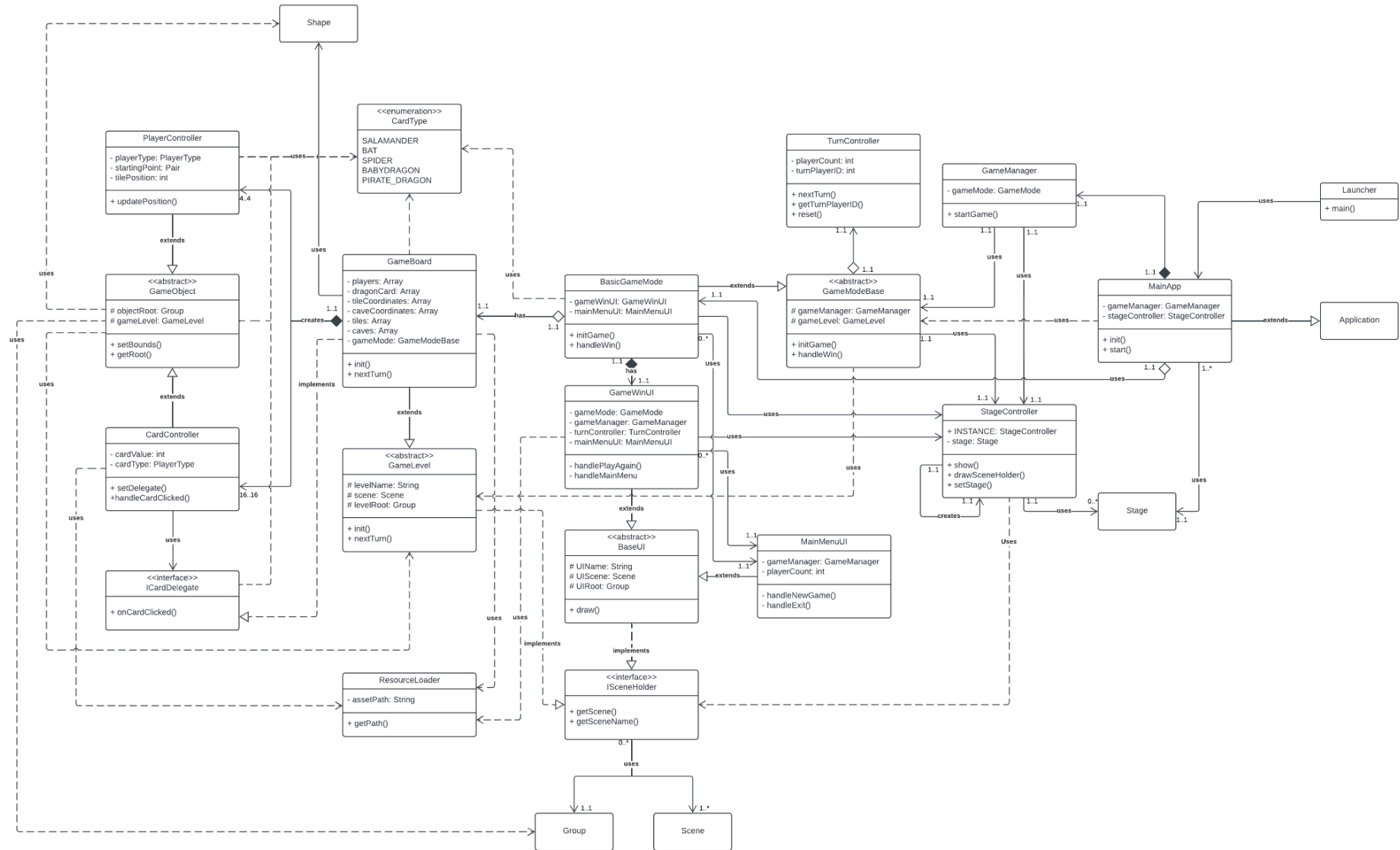
In our final design, we decided against this approach, and instead split it into two classes to achieve separation of concerns. This ensures that each class handles its responsibilities independently, preventing unintended interactions between turn management and player information. This separation also improves maintainability, as the logic for handling turns and managing player information is isolated within their respective classes.

GameBoard	GameLevel
Responsibilities	Collaborators
Initializes the game	GameObject<<abstract>>
draws game objects	GameObject<<abstract>>
checks the value of the flipped chit card and the subsequent appropriate action the player's token ought to make	TokenController
moves the player's token	TokenController
determines the winner	
displays the winner of the game	GameWinUI
displays the scene that shows the game	ISceneHolder<<interface>>
knows coordinates of each Cave	
know coordinates of each Tile	
invokes the next turn	TokenController

In the alternative design, we had GameBoard take on all the responsibilities of GameBoard, BasicGameMode, and StageController seen above. In this design, the responsibilities of starting the game with the provided settings, drawing the game objects, checking the value of the flipped chit card, handling the token movement, determining the winner, and changing the scene of the stage are all handled by the same class, potentially turning it into a "god class."

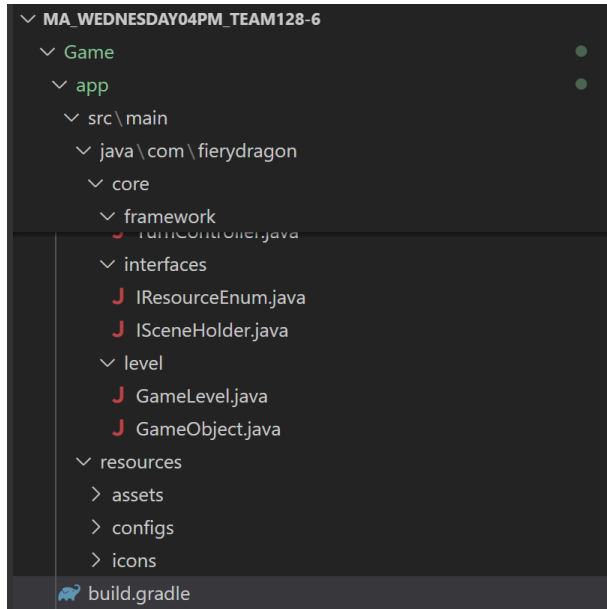
As such, we decided to split these functionalities to separate the concerns of each class, ensuring that each class only has a single responsibility, which improves modularity and maintainability, this ensures that there is no 'god class' that does everything by itself.

Class Diagram



Instruction on how to build the EXE

IMPORTANT NOTE: Windows Operating system and Java 17 is required to build the EXE, if you have another version of Java instead, navigate to build.gradle



Locate this line of code, change 17 to the version number of Java you have installed on your computer.

```
23  ✓ java {  
24  ✓   |   toolchain {  
25  |   |   ..... languageVersion = JavaLanguageVersion.of(17)  
26  |   |   }  
27  |   }
```

To check the Java version you have installed, you can run this command in the terminal: `java -version`

```
PS C:\Users\limas\OneDrive\Desktop\FIT3077\MA_Wednesday04pm_Team128-6> java -version
```

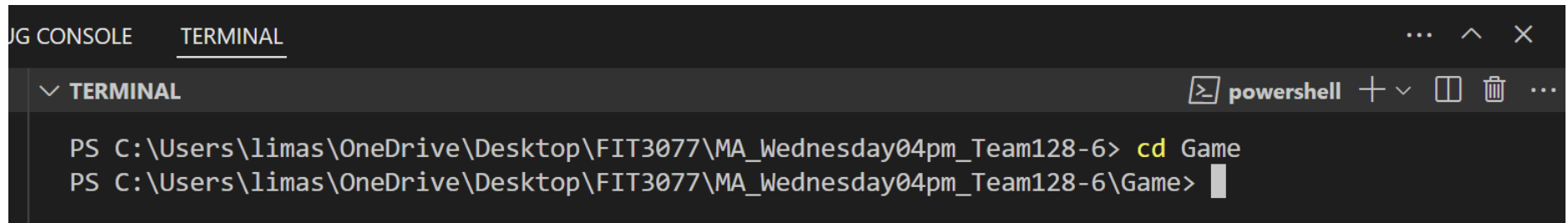
And it will show which version of Java you have installed on your computer.

For example: if you have Java 19 installed on your computer instead of Java 17, change the 17 to 19:

```
23  ✓ java {  
24  ✓   |   toolchain {  
25  |   |   ..... languageVersion = JavaLanguageVersion.of(19)  
26  |   |   }  
27  |   }
```

After that is done, go through the following steps:

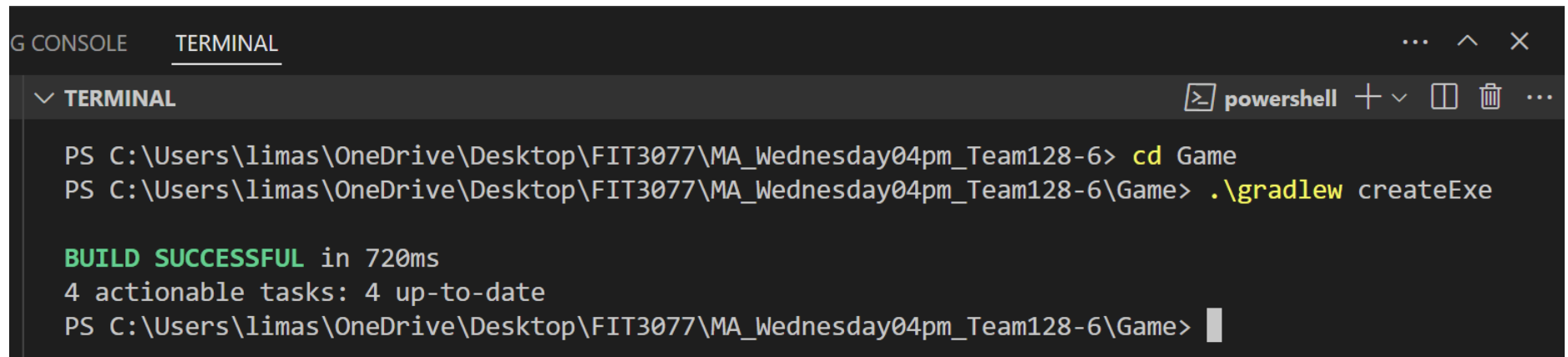
1. In the terminal, navigate to the 'Game' folder



A screenshot of a Visual Studio Code terminal window. The title bar shows 'JG CONSOLE' and 'TERMINAL'. The terminal content shows the user navigating to the 'Game' folder using the 'cd' command. The prompt is 'PS C:\Users\limas\OneDrive\Desktop\FIT3077\MA_Wednesday04pm_Team128-6>' and the command entered is 'cd Game'. The next line shows the prompt has moved to 'PS C:\Users\limas\OneDrive\Desktop\FIT3077\MA_Wednesday04pm_Team128-6\Game>'.

```
PS C:\Users\limas\OneDrive\Desktop\FIT3077\MA_Wednesday04pm_Team128-6> cd Game
PS C:\Users\limas\OneDrive\Desktop\FIT3077\MA_Wednesday04pm_Team128-6\Game>
```

2. Type this command: `.\gradlew createExe`





A screenshot of a Visual Studio Code terminal window. The title bar shows 'JG CONSOLE' and 'TERMINAL'. The terminal content shows the user running the 'gradlew createExe' command. The prompt is 'PS C:\Users\limas\OneDrive\Desktop\FIT3077\MA_Wednesday04pm_Team128-6>' and the command entered is 'cd Game'. The next line shows the prompt has moved to 'PS C:\Users\limas\OneDrive\Desktop\FIT3077\MA_Wednesday04pm_Team128-6\Game>' and the command entered is '.\gradlew createExe'. The output shows 'BUILD SUCCESSFUL in 720ms' and '4 actionable tasks: 4 up-to-date'. The final line shows the prompt has moved back to 'PS C:\Users\limas\OneDrive\Desktop\FIT3077\MA_Wednesday04pm_Team128-6\Game>'.

```
PS C:\Users\limas\OneDrive\Desktop\FIT3077\MA_Wednesday04pm_Team128-6> cd Game
PS C:\Users\limas\OneDrive\Desktop\FIT3077\MA_Wednesday04pm_Team128-6\Game> .\gradlew createExe

BUILD SUCCESSFUL in 720ms
4 actionable tasks: 4 up-to-date
PS C:\Users\limas\OneDrive\Desktop\FIT3077\MA_Wednesday04pm_Team128-6\Game>
```


- The EXE will be created, and can be located by going to Game>app>build>launch4j, note: 'build' folder will only appear after steps 1 and 2 have been completed.

FIT3077 > MA_Wednesday04pm_Team128-6 > Game > app > build > launch4j >				
↑↓ Sort ▾ ≡ View ▾ ⋮				
Name	Date modified	Type	Size	
 lib	19/5/2024 10:28 PM	File folder		
 FieryDragon.exe	19/5/2024 10:28 PM	Application	12,144 KB	

4. To run the EXE, double click on it. The game window will appear, maximize the window to enjoy the full experience

