Umm!! Aroiii Documentation

Created by

Sahutsawat Sivakosit 6231367921

Pongsapak Pulthasthan 6231340921

2110215 Programming Methodology

Semester 2 Year 2019

Chulalongkorn University

Umm!! Aroiii

1. Introduction

Umm!! Aroiii is inspired from the game name “Overcooked 2”. In the game, you are assigned as a head chef in a poor restaurant. Your duty is to serve a food to please customers to make them your regular customers. This game’s objective is to send orders as ordered to get the score as high as possible.

2. Game Controls

Main Menu is the first page of the game. This page contains two buttons that are “Play” and “Quit” buttons. “Play” button will lead you to the next two buttons. To end the game, click “Quit” button

Figure 1: Start screen

After you have clicked the “Play” button, you can choose which mode you want to play. That is as a single player by clicking “1 Player” button or as a duo player by clicking “2 Player” button. By the ways, you can also end the game by clicking the “Quit” button.

Figure 2: Start screen after having pressed play button

When the game starts, you can control your player 1 character by pressing “A” (Left), “W” (Up) , “D” (Right) or “S” (Down) on the keyboard while you can control your player 2 character by pressing “J”(Left), “I”(Up), “L”(Right), “K”(Down).

In order to pick or place a food or put the food in the dish, press the “LSHIFT” in player 1 and press the “:” in player 2. The last button is for cooking. To simply said, it is used to cook ingredients by pressing “LCTRL” in player 1, and pressing “ “ ” in player 2. If you are confused with the button above, please see the next page.

Game Screen

Figure 3: Game Screen

Figure 4: Player 1's character

Figure 5: Player 2's character

There are three types of ingredients in this game: Cabbage, Tomato, and Fish.

Figure 6: Ingredients

Each of the ingredients can be cooked differently. Both tomato and cabbage can only be sliced in the cutting board while fish can both be sliced and fried.

Note: When an ingredient has already been cooked, it can't be cooked anymore.

Figure 7: Sliced tomato

Figure 8: Sliced cabbage

Figure 9: Sliced fish

Figure 10: Fried fish

In this game, the equipment to cook these ingredients is a cutting board and a frying pan. If you want to slice it, go to a cutting board. In contrast, if you want to fry it, go to a frying pan.

Note: You can't place an ingredient if you interact the wrong equipment.

Note: When a player is holding an ingredient, he has to place that ingredient before doing anything such as cooking or picking a new ingredient.

Figure 11: Frying pan

Figure 12: Player is holding a tomato

Figure 13: Cutting board

When you are cooking the ingredient, it will show the progress bar to show your remaining time. The white color represents remaining time.

Figure 14: cooking

During the game, you can pick a new item by going to its storage and interact it.

Note: a player's hand has to be empty so that he can pick a new item.

Figure 15: Item storage

One more thing that is important in this game is dishes. After you have cooked the ingredients, you have to place it in the dish by clicking "LSHIFT" button

Note: You can't put an uncooked ingredient in a dish.

Note: You can't put the cooked ingredient in a dish if you have that ingredient on the dish.

Figure 16: Empty dish

Figure 17: Player is holding a dish filled with a tomato

Other entity blocks in this game are food counter, obstacle, station and bin

- Food counter is a place to send the order. If you send the wrong dish, you will get the point penalty. That is minus five points from player's score. If you send it as ordered, you will get the score depends on the type of the dish.
- Obstacle is a block that will block you from passing through.
- Station is a block which you can place and pick ingredients or dish.

Note: The cooked ingredients can't be placed in stations; you have to put a cooked ingredient on a dish. After that, you can place it on the station.

- Bin is a block which you can dispose of what you are holding.

Figure 18: Sliced fish on a station

Figure 19: Bin

Figure 20: Food counter

Figure 21: Obstacle

Figure 22: Order pane

This is the order pane that is on top of the screen. This is where the orders will be ordered every 20 seconds if you play as a single player and will be ordered every 16 seconds if you play as a duo player. You have to send the order as ordered in this pane from left to right. The maximum order in this game is 5 orders. If the order is left to zero, it will create one more order instantly.

The menus have 3 dishes. They are sashimi salad, simple salad and fried fish. These menus use different ingredients, so you can see what it uses in the menus. If the ingredient has a chopping knife below, it has to be cooked in a cutting board. If the ingredient has a frying pan below, it has to be cooked in a frying pan.

Figure 23: Sashimi salad

Figure 24: Simple salad

Figure 25: Fried fish

Later, this is the order box that is contained in order pane.

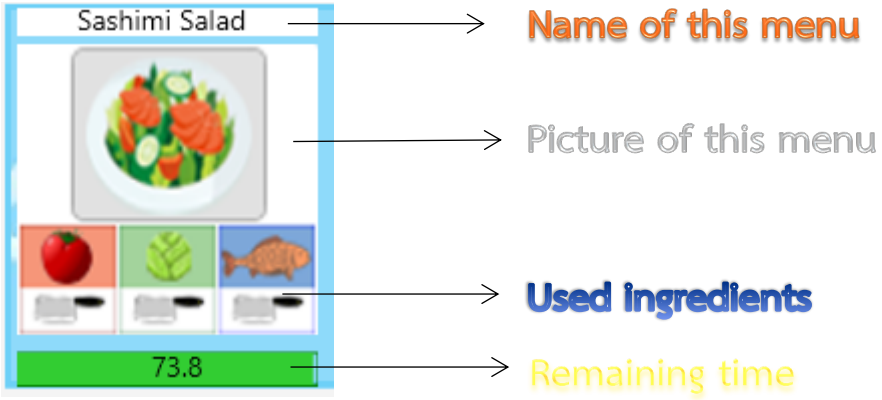


Figure 26: Order box

Next is the timer at the top right of the screen. This is the timer that will be decreased every second. The game will end if the timer reaches 0.

Figure 27: Timer

The last but not least is the data pane. It is located below the screen. This pane provides all of the information in this game. The score represents the current score you have.The money represents the current money a player has which is used for buying ingredients. The score can also be negative but money can't. The next three boxes are storages that keep your remaining ingredients. If the number of it is 0, you have to buy it in the shop so that you can pick it.

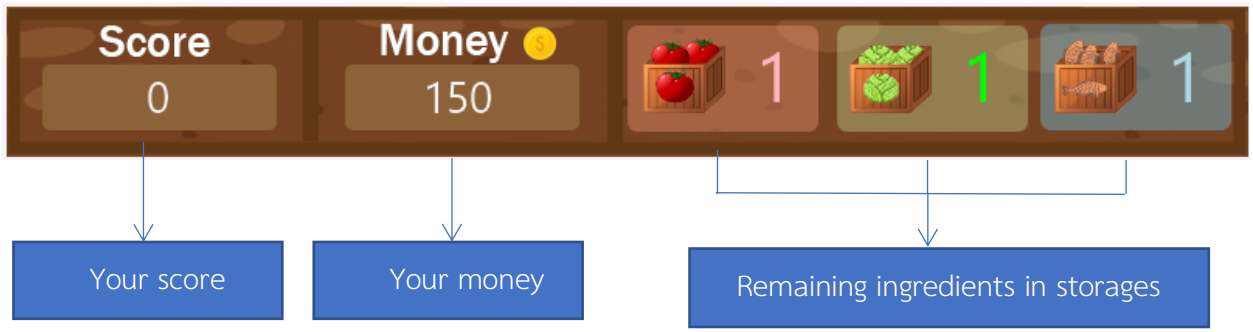


Figure 28: Data pane

Figure 29: The ingredient boxes when the number of ingredient is 0

The final part in the screen is the shop pane that is in the center right of the screen. This part performs buying ingredients. You can see that each ingredient has a different price, and you can buy it by clicking “+” button to increase the amount and decrease the amount of it by clicking “-” button. After you have selected all of the ingredients you want to buy, press the button “Buy” to buy it. It will automatically increase the amount of its ingredient in the storage in the data pane. If the price exceeds your current money, the color of it will be grey. If it can be bought, the color of the button will be green.

Figure 30: default shop

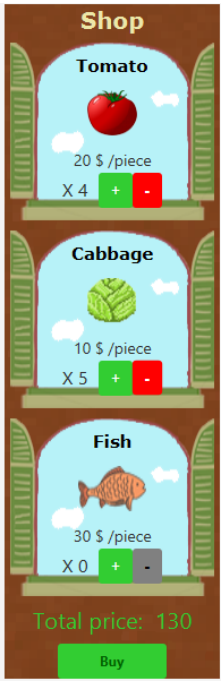


Figure 31: price <= current money

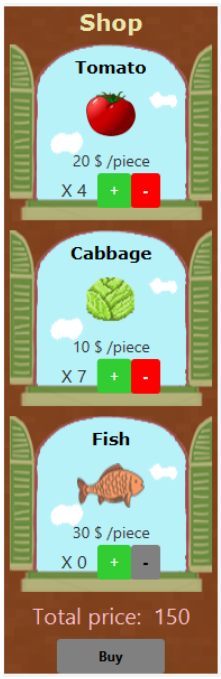


Figure 32: price > money

End Screen

This screen has 2 Screens which depends on the score a player has. This page reports all of the dishes that we sent including successful and failed dish, score, money bonus and total score. Money bonus is calculated by the money you have divided by 10. You can exit the game by clicking the “Quit” button.

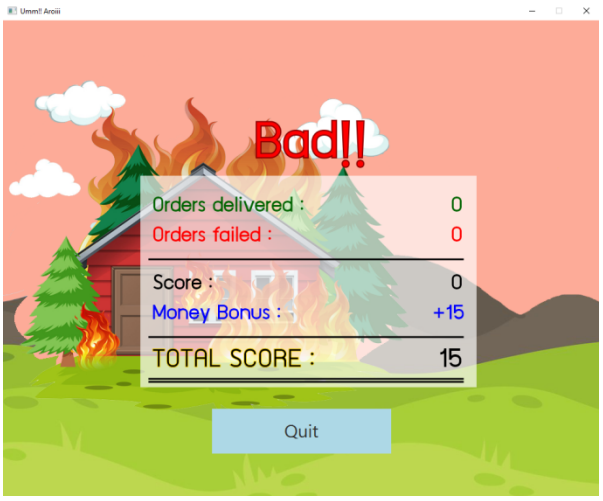


Figure 33: Bad end screen

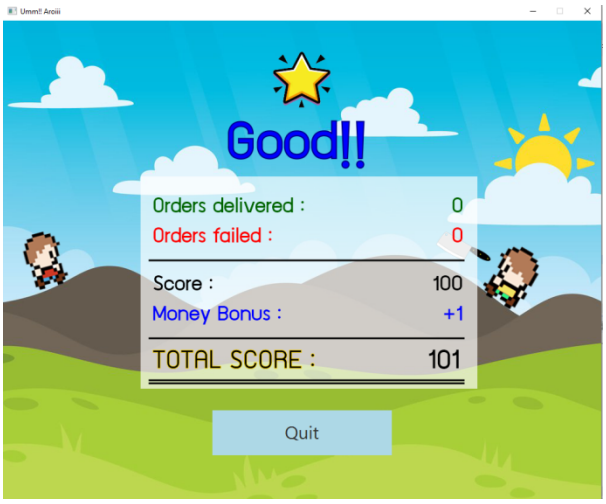


Figure 34: Good end screen

Figure 35: Very good end screen

Figure 36: Excellent end screen

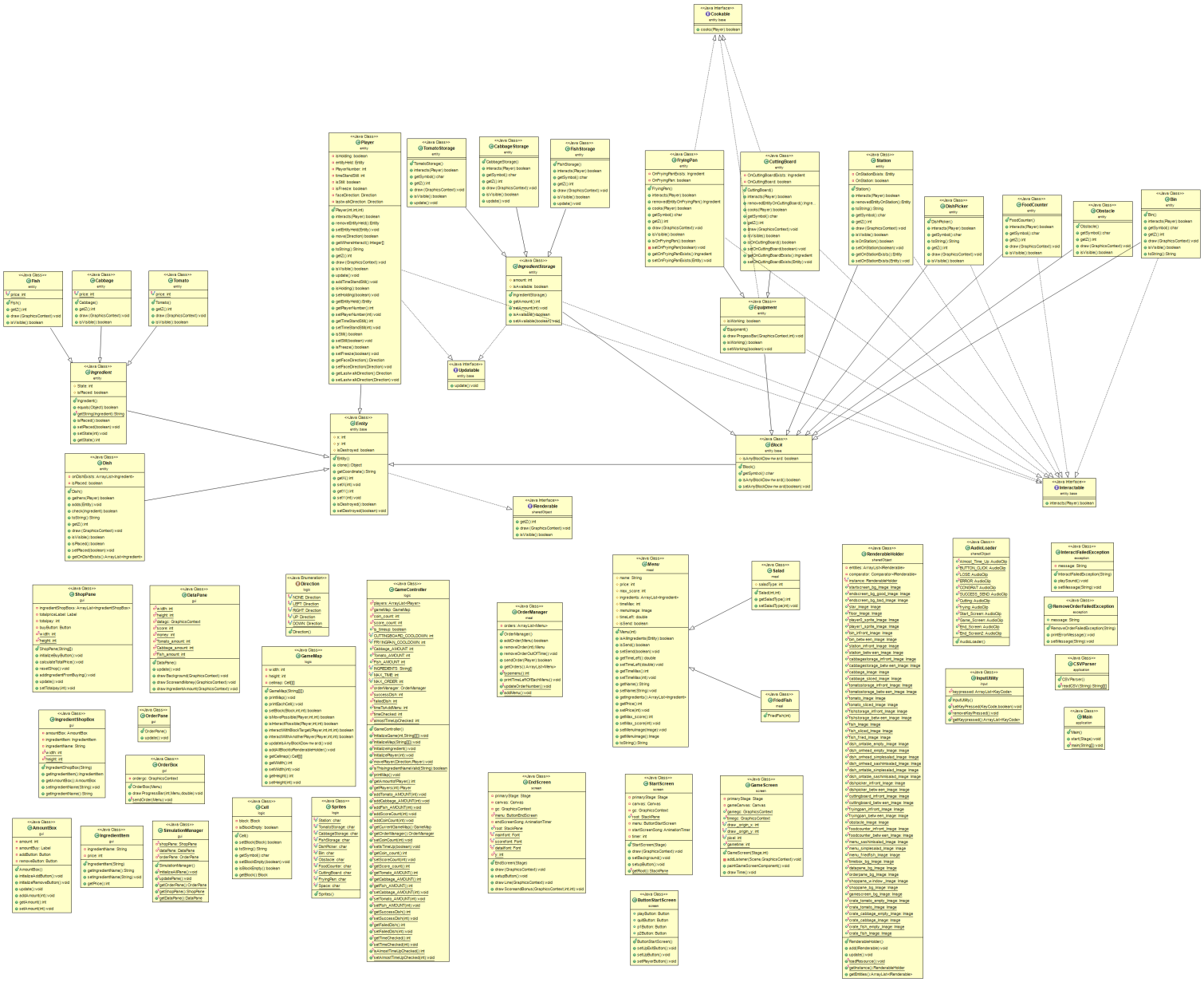
3. Key Controls

Key	Explanation
W	Move your player 1 character up.
A	Move your player 1 character left.
S	Move your player 1 character down.
D	Move your player 1 character right.
LSHIFT	Pick and place items for player 1.
LCTRL	Cook ingredients for player 1.
I	Move your player 2 character up.
J	Move your player 2 character left.
K	Move your player 2 character down.
L	Move your player 2 character right.
:	Pick and place items for player 2.
“	Cook ingredients for player 2.

Green represents player 1

Red represents player 2

4. UML diagrams



5. Class Details

1. Package application

1.1 class CSVParser

This class is used to open and read csv files which containing codes with represent for each block in gamemap.

1.1.1 Method

Name	Description
+ String[ ][ ] readCSV(String filename)	- Receive filename.csv which is a map file in csv folder (in res folder) - read csv file and then return String[ ][ ] containing codes which represent for each block in a gamemap.

1.2 class Main extends Application

1.2.1 Method

Name	Description
+ void start(Stage primaryStage)	- launch up the new StartScreen of the game - show the primaryStage
+ void main(String[ ] args)	the entry point of the application

2. Package entity

2.1 class Bin extends Block implements Interactable

This class represents the Bin block which player can throw anything that he/she doesn't want it anymore into it.

2.1.1 Method

Name	Description
+ boolean interacts(Player p) throws InteractFailedException	Check if player is holding anything in his/her hands? If yes, remove entity in his/her hands. Otherwise throw new InteractFailedException with message "ERROR".
+ char getSymbol()	return Sprites.Bin
+ int getZ()	return getY()*3
+ void draw (GraphicsContext gc)	Draw the Bin block on its current position on the GameScreen.
+ boolean isVisible()	Since the bin cannot be destroyed from the game, it always returns true.
+ String toString()	Return String which represents some details about this Bin.

2.2 class Cabbage extends Ingredient

This class represents Cabbage entity which is the ingredient that can be cooked.

2.2.1 Field

Name	Description
+ final int price	Price of the one piece of cabbage = 10

2.2.2 Constructor

Name	Description
+ Cabbage()	Initialize fields (state = 0) and setPlaced to false

2.2.3 Method

Name	Description
+ int getZ()	return getY()*3+2
+ void draw (GraphicsContext gc)	If the cabbage is not placed, draw the cabbage on its current position on the GameScreen.
+ boolean isVisible()	If the cabbage is not destroyed, return true. Otherwise, return false.

2.3 class CabbageStorage extends IngredientStorage

This class represents CabbageStorage block which player come to pick up new cabbage at this block.

2.3.1 Method

Name	Description
+ boolean interacts(Player p) throws InteractFailedException	Check if this storage is still available (there sill cabbages left) and the player is not holding any item in his/her hands, then the player should pick a new Cabbage, and return true. Otherwise throw new InteractFailedException with message “ERROR”.
+ char getSymbol()	return Sprites.CabbageStorage
+ int getZ()	return getY()*3
+ void draw (GraphicsContext gc)	Draw the CabbageStorage block on its current position on the GameScreen.

+ boolean isVisible()	Since the CabbageStorage cannot be destroyed from the game, it always returns true.
+ void update()	- Setting amount of the cabbage got from GameController - If the amount less than or equal to 0, set available to false. Otherwise, set available to true.

2.4 class CuttingBoard extends Equipment implements Interactable, Cookable

This class represents CuttingBoard equipment where player can use this to chop the cabbage, tomato and fish.

2.4.1 Field

Name	Description
- Ingredient OnCuttingBoardExists	The ingredient which is placed on the CuttingBoard.
- boolean OnCuttingBoard	State that is anything is placed on the CuttingBoard.

2.4.2 Constructor

Name	Description
+ CuttingBoard()	Initialize these fields - setOnCuttingBoardExists to null - setOnCuttingBoard to false - set that this equipment is not working.

2.4.3 Method

Name	Description
+ boolean interacts(Player p) throws InteractFailedException	- Check that if player is not holding anything, and there is a ingredient placed on the CuttingBoard, then the player should pick up that ingredient and return true.

	- If the player is holding a dish, there is a cooked ingredient placed on the CuttingBoard, and the type of ingredient is not repeat with the ingredients on the dish, then the player should put that ingredient into the dish and return true. - If the player is holding an ingredient, and there is nothing placed on the CuttingBoard, then the player should place down that ingredient on the CuttingBoard and return true. - Otherwise, throw new InteractFailedException with message “ERROR”.
+ Ingredient removeEntityOnCuttingBoard()	Removing ingredient placed on the CuttingBoard by set destroyed to true and return the removed ingredient.
+ boolean cooks(Player p) throws InteractFailedException	- If the CuttingBoard is not being used by the other player, and the ingredient on the board hasn’t been cooked yet, start chopping that ingredient (and also drawing progress bar with AnimationTimer), and return true. - If there is nothing placed on the CuttingBoard or the player is holding something or the ingredient placed on the CuttingBoard is already cooked, throw new InteractFailedException with message “ERROR”.
+ char getSymbol()	return Sprites.CuttingBoard
+ int getZ()	return getY()*3
+ void draw (GraphicsContext gc)	Draw the CuttingBoard block and the ingredient placed on the CuttingBoard on its current position on the GameScreen.
+ boolean isVisible()	Since the CuttingBoard cannot be destroyed from the game, it always returns true.
+ GETTER & SETTER for each field.	

2.5 class Dish extends Entity

This class represents Dish Entity which is used to contain our cooked ingredient, and then served to the customers.

2.5.1 Field

Name	Description
- ArrayList<Ingredient> OnDishExists	ArrayList contains ingredients on the dish.
- boolean isPlaced	State that the dish is placed on the station or not.

2.5.2 Constructor

Name	Description
+ Dish()	Initialize these fields - setPlaced and setDestroyed to false - initialize the OndishExists with new ArrayList.

2.5.3 Method

Name	Description
+ boolean gathers(Player p)	If the player is holding cooked ingredient in his/her hands, and the dish hasn't contained that ingredient yet, then add that ingredient into the dish, and return true. Otherwise, return false.
+ void adds(Entity e)	If entity is an ingredient, add that ingredient into the dish.
+ boolean check(Ingredient i)	This method check that is the ingredient can be added to the dish or not.
+ String toString()	Return String which represents some details about this Dish.
+ int getZ()	return getY()*3+2

+ void draw (GraphicsContext gc)	Draw the Dish and the ingredient placed on the dish on its current position on the GameScreen.
+ boolean isVisible()	If the dish is not destroyed, return true. Otherwise, return false.
+ GETTER & SETTER for isPlaced	
+ GETTER for OnDishExists	

2.6 class DishPicker extends Block implements Interactable

This class represents DishPicker Block where the player can pick the new dish.

2.6.1 Method

Name	Description
+ boolean interacts(Player p) throws InteractFailedException	Check if the player is not holding anything, then the player should pick a new dish, and return true. Otherwise, throw new InteractFailedException with message “ERROR”.
+ char getSymbol()	return Sprites.DishPicker
+ String toString()	Return String which represents some details about this DishPicker.
+ int getZ()	return getY()*3
+ void draw (GraphicsContext gc)	Draw the DishPicker block on its current position on the GameScreen.
+ boolean isVisible()	Since the DishPicker cannot be destroyed from the game, it always returns true.

2.7 abstract class Equipment extends Block implements Interactable, Cookable

2.7.1 Field

Name	Description
# boolean isWorking	State that this equipment is being used or not.

2.7.2 Method

Name	Description
+ void drawProgressBar(GraphicsContext gc, int maxTime)	This method draws a progress bar above the equipment on the screen, when the player is using this equipment to cook the ingredient.
+ GETTER & SETTER for isWorking	

2.8 class Fish extends Ingredient

This class represents Fish entity which is the ingredient that can be cooked.

2.8.1 Field

Name	Description
+ final int price	Price of the one piece of fish = 30

2.8.2 Constructor

Name	Description
+ Fish()	Initialize fields (state = 0) and setPlaced to false

2.8.3 Method

Name	Description
+ int getZ()	return getY()*3+2

+ void draw (GraphicsContext gc)	If the fish is not placed, draw the cabbage on its current position on the GameScreen.
+ boolean isVisible()	If the fish is not destroyed, return true. Otherwise, return false.

2.9 class **FishStorage** extends IngredientStorage

This class represents FishStorage block which player come to pick up new fish at this block.

2.9.1 Method

Name	Description
+ boolean interacts(Player p) throws InteractFailedException	Check if this storage is still available (there sill fish left) and the player is not holding any item in his/her hands, then the player should pick a new Fish, and return true. Otherwise throw new InteractFailedException with message “ERROR”.
+ char getSymbol()	return Sprites.FishStorage
+ int getZ()	return getY()*3
+ void draw (GraphicsContext gc)	Draw the FishStorage block on its current position on the GameScreen.
+ boolean isVisible()	Since the FishStorage cannot be destroyed from the game, it always returns true.
+ void update()	- Setting amount of the fish got from GameController - If the amount less than or equal to 0, set available to false. Otherwise, set available to true.

2.10 class FoodCounter extends Block implements Interactable

This class represents FoodCounter block which player can serve dishes to the customers here.

2.10.1 Method

Name	Description
+ boolean interacts(Player p) throws InteractFailedException	- If the player is not holding a dish in his/her hands, throw new InteractFailedException with message "There is nothing to be delivered". - Otherwise, send the dish being held by the player to the OrderManager, if the menu is in the list, return true. Otherwise, throw new InteractFailedException with message "The carried menu isn't in the list".
+ char getSymbol()	return Sprites.FoodCounter
+ int getZ()	return getY()*3
+ void draw (GraphicsContext gc)	Draw the FoodCounteer block on its current position on the GameScreen.
+ boolean isVisible()	Since the FoodCounter cannot be destroyed from the game, it always returns true.

2.11 class FryingPan extends Equipment implements Interactable, Cookable

This class represents FryingPan equipment where player can use this to fry fish.

2.11.1 Field

Name	Description
- Ingredient OnFryingPanExists	The ingredient which is placed on the FryingPan.
- boolean OnFryingPan	State that is anything is placed on the FryingPan.

2.11.2 Constructor

Name	Description
+ FryingPan()	Initialize these fields - setOnFryingPanExists to null - setOnFryingPan to false - set that this equipment is not working.

2.11.3 Method

Name	Description
+ boolean interacts(Player p) throws InteractFailedException	- Check that if player is not holding anything, and there is a ingredient placed on the FryingPan, then the player should pick up that ingredient and return true. - If the player is holding a dish, there is a cooked ingredient placed on the FryingPan, and the type of ingredient is not repeat with the ingredients on the dish, then the player should put that ingredient into the dish and return true. - If the player is holding raw fish or fried fish, and there is nothing placed on the FryingPan, then the player should place down that fish on the FryingPan and return true. - Otherwise, throw new InteractFailedException with message “ERROR”.
+ Ingredient removeEntityOnFryingPan()	Removing ingredient placed on the FryingPan by set destroyed to true and return the removed ingredient.
+ boolean cooks(Player p) throws InteractFailedException	- If the FryingPan is not being used by the other player, and the ingredient on the board hasn’t been cooked yet, start frying that ingredient (and also drawing progress bar with AnimationTimer), and return true.

	- If there is nothing placed on the FryingPan or the player is holding something or the ingredient placed on the FryingPan is already cooked, throw new InteractFailedException with message “ERROR”.
+ char getSymbol()	return Sprites.FryingPan
+ int getZ()	return getY()*3
+ void draw (GraphicsContext gc)	Draw the FryingPan block and the ingredient placed on the FryingPan on its current position on the GameScreen.
+ boolean isVisible()	Since the FryingPan cannot be destroyed from the game, it always returns true.
+ GETTER & SETTER for each field.	

2.12 abstract class Ingredient extends Entity

This class provides some information about the ingredient.

2.12.1 Field

Name	Description
# int State	State that the ingredient is raw or cooked. (0 = raw, 1 = sliced, 2 = fried)
# boolean isPlaced	State that the ingredient is placed on the station/ equipment or not.

2.12.2 Method

Name	Description
+ boolean equals(Object obj)	Compare that the ingredient and the object is at same type and same state or not.
+ String getString(Ingredient i)	Return a String which tell the ingredient type.
+ GETTER & SETTER for each field.	

2.13 abstract class IngredientStorage extends Block implements Interactable, Updatable

This class provides some information about the ingredient storage.

2.13.1 Field

Name	Description
# int amount	The amount of the ingredient left in the storage.
# boolean isAvailable	State that there is any ingredient left in the storage or not.

2.13.2 Method

Name	Description
+ GETTER & SETTER for each field	

2.14 class Obstacle extends Block

This class represents Obstacle Block where the player isn't allowed to walk through this block. The player can't do anything with this block, just have to walk to the other way when he/she get stuck.

2.14.1 Method

Name	Description
+ char getSymbol()	return Sprites.Obstacle
+ int getZ()	return getY()*3
+ void draw (GraphicsContext gc)	Draw the Obstacle block on its current position on the GameScreen.
+ boolean isVisible()	Since the Obstacle cannot be destroyed from the game, it always returns true.

2.15 class Player extends Entity implements Interactable, Updatable

This class represents Player Entity whom you are controlling, the player is the main character of the game, he/she can interact with many blocks, such as DishPicker, Bin, and IngredientStorage, and he/she is able to use the equipment to cook the ingredient.

2.15.1 Field

Name	Description
- boolean isHolding	State that is the player holding anything in his/her hands.
- Entity entityHeld	The entity which the player is holding.
- int PlayerNumber	The player's number. (Can be only 0 or 1)
- int timeStandStill	The duration of time that the player just stands still and doesn't move.
- boolean isStill	State that the player is standing still for a duration of time or not.
- boolean isFreeze	State that the player is being freeze or not. (Player got freeze, when he/she is cooking with equipment)
- Direction faceDirection	The direction that player is facing toward.
- Direction lastwalkDirection	The direction that player is moving toward.

2.15.2 Constructor

Name	Description
+ Player(int playerNumber, int x, int y)	Initialize the player's number, and (x,y) where the player spawn in a GameMap. - setHolding to false and setEntityHeld to null - setFaceDirection and setLastWalkDirection with Direction.NONE - setTimeStandStill to 0 - setStill and setFreeze to false

2.15.3 Method

Name	Description
+ boolean interacts(Player p) throws InteractFailedException	- If one player is holding an item in his/her hands, and the another player's hands are free, then the one whose hands are empty should get the item which being held by the another player, and return true. - Otherwise, throw new InteractFailedException with message "ERROR". Note: Players are able to interact with each other only in 2 players mode.
+ Entity removeEntityHeld()	Removing item which being held (setDestroyed to true), setHolding to false, and then return the removed item.
+ void setEntityHeld(Entity entityheld)	Setting entity in the player's hands.
+ boolean move(Direction dir)	If the player is not being freeze, and it is able to move to the target position, then reset the player's position to the target position and also the item in his/her hands and return true. Otherwise return false.
+ Integer[ ] getWhereInteract()	This method return back the Interger[ ] containing position (targetx, targety) where the player is going to interact.
+ String toString()	Return String which represents some details about this Player.
+ int getZ()	return getY()*3+1
+ void draw (GraphicsContext gc)	Draw the Player standing or walking on his/her current position on the GameScreen.
+ boolean isVisible()	Since the Player cannot be destroyed from the game, it always returns true.
+ void update()	- If player is standing still for more than a duration of time, then setStill to true, Otherwise setStill to false. - Get keypressed from InputUtility

	- If the player pressed moved key, then move the player, but if the player didn't pressed any moved key, then addtimeStill(). - If the player pressed interact or cook key, then get the interact position, then see the result of interacting (or cooking).
+ void addTimeStandStill()	Adding timeStandStill by 1
+ GETTER for EntityHeld	
+ GETTER & SETTER for the other fields.	

2.16 class Station extends Block implements Interactable

This class represents Station Block which is the most important block in the game, because the players have to place dishes and ingredients on the station, while they are cooking.

2.16.1 Field

Name	Description
- Entity OnStationExists	The entities which are placed on the station.
- boolean OnStation	State that is anything is placed on the station.

2.16.2 Constructor

Name	Description
+ Station()	Initialize fields setOnstationExists to null, and setOnStation to false.

2.16.3 Method

Name	Description
+ boolean interacts(Player p) throws InteractFailedException	- If the player's hands are empty, and there is any entity placed on the station, then the player should pick up that entity, and return true. - If the player is holding a dish, and there is a cooked ingredient placed on the station or the player is holding a cooked ingredient, and there is a dish on the station, then the player should pick that ingredient into the dish (if possible), and return true. - If the player is holding a dish or an uncooked ingredient, and the station is empty, then the player should place down the item, and return true. - Otherwise, throw new InteractFailedException with message "ERROR".
+ Entity removedEntityOnStation()	Removing entity placed on the Station by set destroyed to true and return the removed entity.
+String toString()	Return String which represents some details about this Station.
+ char getSymbol()	return Sprites.Station
+ int getZ()	return getY()*3
+ void draw (GraphicsContext gc)	Draw the Station block and the entity placed on the Station on its current position on the GameScreen.
+ boolean isVisible()	Since the Station cannot be destroyed from the game, it always returns true.
+ GETTER & SETTER for each field.	

2.17 class Tomato extends Ingredient

This class represents Tomato entity which is the ingredient that can be cooked.

2.17.1 Field

Name	Description
+ final int price	Price of the one piece of tomato = 20

2.17.2 Constructor

Name	Description
+ Tomato()	Initialize fields (state = 0) and setPlaced to false

2.17.3 Method

Name	Description
+ int getZ()	return getY()*3+2
+ void draw (GraphicsContext gc)	If the tomato is not placed, draw the cabbage on its current position on the GameScreen.
+ boolean isVisible()	If the tomato is not destroyed, return true. Otherwise, return false.

2.18 class TomatoStorage extends IngredientStorage

This class represents TomatoStorage block which player come to pick up new tomato at this block.

2.18.1 Method

Name	Description
+ boolean interacts(Player p) throws InteractFailedException	Check if this storage is still available (there sill tomato left) and the player is not holding any item in his/her hands, then the player should pick a new Tomato, and return

	true. Otherwise throw new InteractFailedException with message “ERROR”.
+ char getSymbol()	return Sprites.TomatoStorage
+ int getZ()	return getY()*3
+ void draw (GraphicsContext gc)	Draw the TomatoStorage block on its current position on the GameScreen.
+ boolean isVisible()	Since the TomatoStorage cannot be destroyed from the game, it always returns true.
+ void update()	- Setting amount of the tomato got from GameController - If the amount less than or equal to 0, set available to false. Otherwise, set available to true.

3. Package entity.base

3.1 abstract class **Block** extends Entity

3.1.1 Field

Name	Description
# boolean isAnyBlockDownward	State that is there any block below this block. (Use for rendering images)

3.1.2 Method

Name	Description
+ char getSymbol()	Return a letter code representing for each block in the game.
+ GETTER & SETTER for isAnyBlockDownard	

3.2 interface Cookable

This interface defines method for Entity (Block) that can be used to cook.

3.2.1 Method

Name	Description
+ boolean cooks(Player p) throws InteractFailedException	This method is called when the Player pressed cook button on the keyboard. It returns true if the player is interacting with equipment and the ingredient is ready to be cooked. Otherwise, throw InteractFailedException.

3.3 abstract class **Entity** implements IRenderable, Cloneable

This class provide basic fields for all entities.

3.3.1 Field

Name	Description
# int x	X coordinate of the entity
# int y	Y coordinate of the entity
# boolean isDestroyed	State that the entity has been destroyed or not.

3.3.2 Method

Name	Description
+ Object Clone()	Cloning object and return the cloned object.
+ String getCoordinate()	Return String indicating the coordinate where the entity is.
+ GETTER & SETTER for each field	

3.4 interface Interactable

This interface defines method for Entity that can be interacted with.

3.4.1 Method

Name	Description
+ boolean interacts(Player p) throws InteractFailedException	This method is called when the Player pressed interact button the keyboard. It returns true when the interaction is complete. Otherwise, throw InteractFailedException.

3.5 interface Updatable

This interface defines method for Entity that need to be updated while the game was running.

3.5.1 Method

Name	Description
+ void update()	This method is always called while the game is running. This method helps updating some fields of the entity that implements Updatable.

4. Package exception

4.1 class **InteractFailedException** extends Exception

This exception occurs when the interaction between the player and the entity is not correct or not comply with the game rules.

4.1.1 Field

Name	Description
- String message	Description of the error.

4.1.2 Constructor

Name	Description
+ InteractFailedException(String message)	setMessage of the error to the message field.

4.1.3 Method

Name	Description
+ void playSound()	When this exception was caught by the catch-block, play ERROR sound from the AudioLoader.
+ SETTER for message field	

4.2 class **RemoveOrderFailedException** extends Exception

This exception occurs when the system is not able to remove the order from the OrderManager.

4.2.1 Field

Name	Description
- String message	Description of the error.

4.2.2 Constructor

Name	Description
+ RemoveOrderFailedException(String messge)	setMessage of the error to the message field.

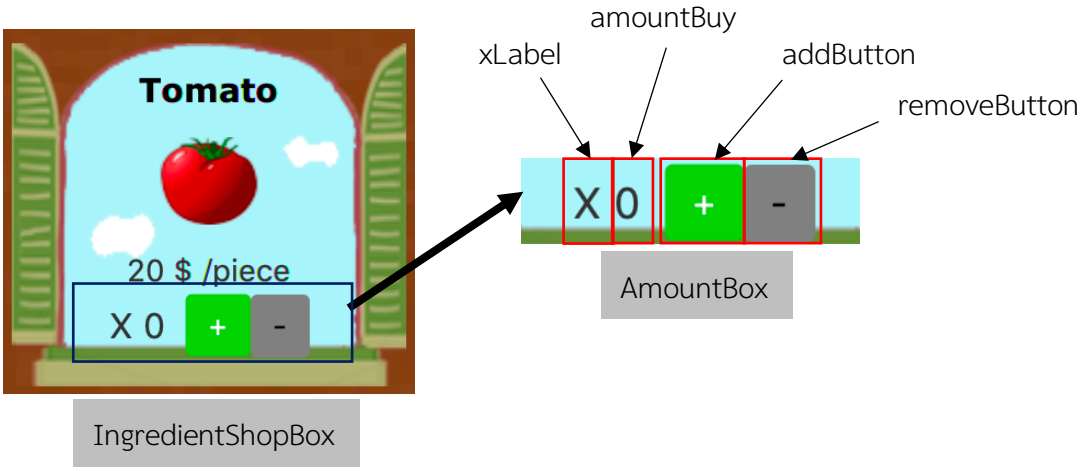
4.2.3 Method

Name	Description
+ void printErrorMessage()	When this exception was caught by the catch-block, print the error message.
+ SETTER for message field	

5. Package gui

5.1 class **AmountBox** extends HBox

This class is the box that player can click to add or remove amount of the ingredient in the IngredientShopBox.



5.1.1 Field

Name	Description
- int amount	Amount of the ingredient that player wants to buy.
- Label amountBuy	The label for displaying the amount of the ingredient that player wants to buy.
- Button addButton	The button for increasing the amount of ingredient.
- Button removeButton	The button for decreasing the amount of ingredient.

5.1.2 Constructor

Name	Description
+ AmountBox()	Constructor method. - Sets amount field to 0. Initialize with the following specifications. - Sets preferred height to 32, preferred width to 128. - Sets the alignment to CENTER.

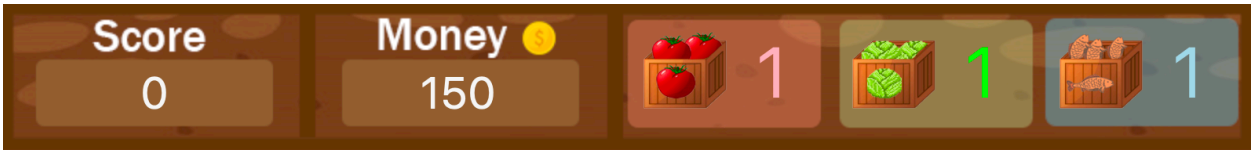
	- Initializes xLabel with “X “ and set its font with size 16. - Initializes amountBuy (Label) with “0 “and set its font with size 16. - Initializes addButton with initializeAddButton() - Initiailizes removeButton with initializeRemoveButton()

5.1.3 Method

Name	Description
+ void initializeAddButton()	- Initializes button with text “+” - Sets preferred with to 16 - Sets onAction to handle by does the following: - Add amount of the ingredient by 1 - Play click sound from AudioLoader
+ void initializeRemoveButton()	- Initializes button with text “-” - Sets preferred with to 16 - Sets onAction to handle by does the following: - Decrease amount of the ingredient by 1 - Play click sound from AudioLoader
+ void update()	This method is used to update button background color.
+ void addAmount(int n)	This method is used to add amount the ingredient that the player wants to buy. (The amount can’t be lower than zero)
+ GETTER & SETTER for amount	

5.2 class DataPane extends Canvas

This class is the canvas that show the score, money and the amount of each ingredient left in the storage.



DataPane

5.2.1 Field

Name	Description
- int width	The width of the canvas = 800
- int height	The height of the canvas = 96
- GraphicsContext datagc	The GraphicsContext of this canvas
- int score	Your current score
- int money	Your current money
- int Tomato_amount	Amount of tomato left in the storage
- int Cabbage_amount	Amount of cabbage left in the storage
- int Fish_amount	Amount of fish left in the storage

5.2.2 Constructor

Name	Description
+ DataPane()	Constructor method. - Sets width and height - Initializes datagc field by getGraphicsContext from this Canvas

5.2.3 Method

Name	Description
+ void update()	Get score, money, ingredient amount from GameController and draw on the Canvas.

	(This method is being called all the time while the game is running.)
+ void drawBackground(Graphics Context gc)	This method used to draw the background image of the DataPane.
+ void drawScoreandMoney(Graphics Context gc)	This method used to draw the score and money on the Canvas.
+ void drawIngredientAmount(Graphics Context gc)	This method used to draw the amount of the ingredient left in the storage on the Canvas.

5.3 class IngredientItem

5.3.1 Field

Name	Description
- String ingredientName	Name (Type) of the ingredient
- int price	Price of the ingredient

5.3.2 Constructor

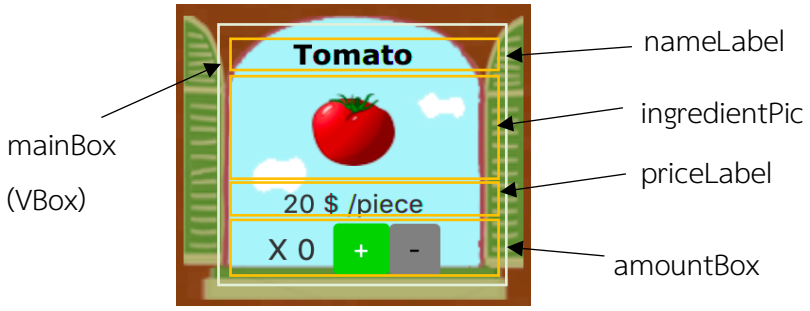
Name	Description
+ IngredientItem(String ingredientName)	If the ingredientName is valid, then initialize the price field.

5.3.3 Method

Name	Description
+ GETTER for price	
+ GETTER & SETTER for ingredientName	

5.4 class **IngredientShopBox** extends StackPane

This class is the pane where the player can see it in the ShopPane, when they want to buy some ingredients.



IngredientShopBox

5.4.1 Field

Name	Description
- AmountBox amountBox	Box where the player can click to add/remove amount of the ingredient bought.
- IngredientItem ingredientItem	Type of the ingredient being sold
- String ingredientName	Name of the ingredient being sold
- int width	The width of the box = 180
- int height	The height of the box = 157

5.4.2 Constructor

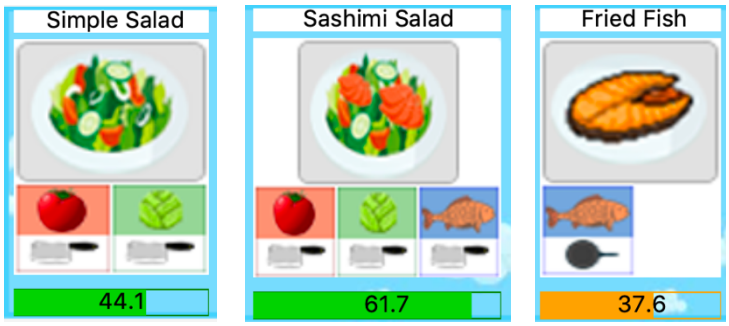
Name	Description
+ IngredientShopBox(String ingredientName)	Initializes the background, nameLabel, picture of ingredient, priceLabel and amountBox.

5.4.3 Method

Name	Description
+ GETTER for each field	
+ SETTER for ingredientName	

5.5 class **OrderBox** extends Canvas

This class is the box where the player can see it in the OrderPane, it tells us want menu the customers wanting us to serve them, and the time limit for each menu.



OrderBox

5.5.1 Field

Name	Description
- GraphicsContext ordergc	The GraphicsContext of this canvas

5.5.2 Constructor

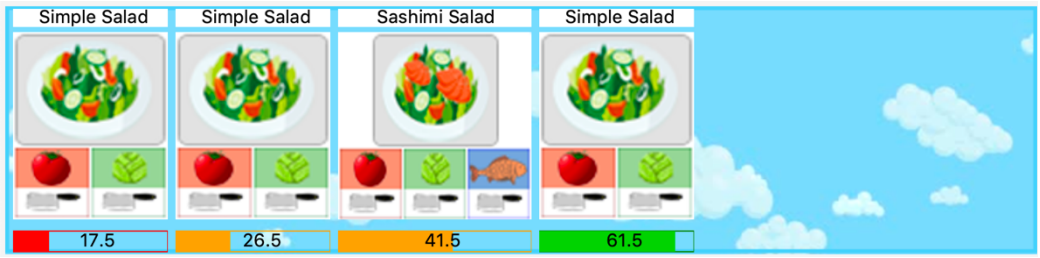
Name	Description
+ OrderBox(Menu menu)	Initializes the fields, draw menu's image, and draw a progress bar below the picture by using drawProgressBar method.

5.5.3 Method

Name	Description
+ drawProgressBar(int maxTime, Menu menu, double Timeleft)	Drawing a progress bar from the time limit left of the menu by using an animationTimer.
+ void sendOrder(Menu menu)	Set the send field of the menu to true.

5.6 class OrderPane extends HBox

This class is the pane where all the order list from the customers are shown, so you are able to see what the menu you have to do next.



OrderPane

5.6.1 Constructor

Name	Description
+ OrderPane()	Constructor method - Sets preferred width to 800, preferred height to 192 - Sets spacing to 6, insets with new Insets(0,0,0,6) - Get all Orders from the OrderManager, create and shown OrderBoxs respectively.

5.6.2 Method

Name	Description
+ void update()	Clear all the orders in the OrderPane, and re-get all the orders from the OrderManager, re-create and shown OrderBoxs respectively. (This method is being called all the time while the game is running.)

5.7 class ShopPane extends StackPane

This class is the pane where all the IngredientShopBox is shown, you can perform buying ingredients here.

5.7.1 Field

Name	Description
- ArrayList<IngredientShopBox> ingredientShopBoxs	List of ingredients sold.
- Label totalpriceLabel	The label for displaying the money that you have to pay for buying ingredients.
- int totalpay	Total price of all ingredients that player wants to buy.
- button buyButton	The button for performing buy action.
- int width	The width of the pane = 190
- int height	The height of the pane = 596

5.7.2 Constructor

Name	Description
+ ShopPane(String[ ] ingredientName)	Constructor method - Sets preferred width and height - Draws background - Initializes ingredientShopBoxs by get ingredient name from ingredientName and create an IngredientShopBox and add it to the ArrayList - Initializes the totalpriceLabel - Initializes the buyButton with initializeBuyButton()

5.7.3 Method

Name	Description
+ void initializeBuyButton()	- Initializes button with text “Buy” - Sets preferred width to 96, height to 30

	- Sets onAction to handle by does the following: - Check if the total price is not overbudget from the money the player has, if yes then perform buying by calling addIngredientFromBuying() and deduct the money. Otherwise, play sound ERROR from AudioLoader.
+ void calculateTotalPrice()	Get the amount of each type of ingredient that players will buy, then calculate the total price and set text to the totalpriceLabel with the current total price.
+ void resetShop()	This method is called when the player had performed buying action successfully, by resetting all amount of each ingredient in the AmountBox to 0.
+ void addIngredientFromBuying()	This method is called when the player clicks at the buyButton and the player has enough money to pay. This method will add the amount of the ingredient in the storage with the amount of ingredient he/she bought.
+ void update()	This method helps in setting style of the buyButton and totalpriceLabel. (This method is being called all the time while the game is running.)
+ SETTER for totalpay	

5.8 class SimulationManager

This class is used to manage all the GUI parts in the system.

5.8.1 Field

Name	Description
- ShopPane shopPane	The ShopPane where the player can buy ingredient.

- DataPane dataPane	The DataPane where the player can see his/her current score, money and ingredient storages.
- OrderPane orderPane	The OrderPane where the player can see all the orders.

5.8.2 Method

Name	Description
+ void initializeAllPane()	This method is called at the beginning of the application. initializes all the fields by creating new one.
+ void updatePane()	Updating all the pane that need to be updated.
+ GETTER for each field	

6. Package input

6.1 class InputUtility

This class helps handling with all input keycode from the player.

6.1.1 Field

Name	Description
- ArrayList<KeyCode> keypressed	List contains all the keyboard input at any moment.

6.1.2 Method

Name	Description
+ void setKeyPressed(KeyCode keyCode, boolean pressed)	Adding keyCode in to the keypressed ArrayList if it is not already pressed.
+ void removeKeyPressed()	Removing all the keyCode in the keypressed ArrayList.
+ GETTER for keypressed	

7. Package logic

7.1 class Cell

This class stores data of blocks and entities for each coordinate in the map.

7.1.1 Field

Name	Description
- Block block	The block located at this cell
- boolean isBlockEmpty	State that this cell contains no block or not.

7.1.2 Constructor

Name	Description
+ Cell()	Inititalizes the field by setBlock to null and set isBlockEmpty to true.

7.1.3 Method

Name	Description
+ boolean setBlock(Block b)	If this block is empty, then set the block field with b and set isBlockEmpty to false and return true. Otherwise, return false.
+ String toString()	Return String which represents some details about this Cell.
+ char getSymbol()	If this block is empty, return ‘O’. Otherwise, return the block’s symbol.
+ GETTER for block	
+ GETTER & SETTER for isBlockEmpty	

7.2 enum Direction

7.2.1 enum

This enum represents direction. It contains the following values: LEFT, RIGHT, UP, DOWN and NONE.

7.3 class GameController

This class is the game system. Most of the game's global variable are kept here.

7.3.1 Field

Name	Description
- ArrayList<Player> players	List containing of players in the game
- GameMap gameMap	The current map of the game
- int coin_count	Your current money
- int score_count	Your current score
+ boolean is_timeup	State that the time is already run out or not.
+ final int CUTTINGBOARD_COOLDOWN	The time interval takes to chop the ingredient = 2
+ final int FRYINGPAN_COOLDOWN	The time interval takes to fry the ingredient = 5
+ int Cabbage_AMOUNT	The amount of cabbage left in the storage
+ int Tomato_AMOUNT	The amount of tomato left in the storage
+ int Fish_AMOUNT	The amount of fish left in the storage
+ final String[ ] INGREDIENTS	Array containing of all names of the ingredient in the game = ["Cabbage","Tomato","Fish"]
+ final int MAX_TIME	The time it takes to play the game from start to finish = 210
+ final int MAX_ORDER	The max number of orders that can be kept in the orders list at the moment of time = 5
- OrderManager orderManager	The OrderManager which deals with the orders.

- int successDish	The amount of dish you have successfully served.
- int failedDish	The amount of dish you didn't serve in time.
+ int timetoAddMenu	The time interval for the next order.
- int timeChecked	The boolean trigger used for adding menu.
- int almostTimeUpChecked	State that is the time going to run out yet. (Last 20 second of the game)

7.3.2 Method

Name	Description
+ void InitializeGame(int numberOfPlayers, String[ ][ ] map)	Initialize the game. (map and the other important fields)
+ void InitializeMap(String[ ][ ] map)	Initialize the GameMap.
+ void InitializeIngredient()	Initialize the ingredient amount fields.
+ void InitializePlayer(int numberOfPlayers)	Initialize the players.
+ void movePlayer(Direction dir, Player p)	Move the player p with the direction given.
+ boolean isThisIngredientNameValid(String ingredientName)	Check that is the ingredient name inserted is valid in this game or not.
+ void printMap()	Print out the current map.
+ int getAmountofPlayer()	Return the amount of player.
+ Player getPlayers(int numberPlayer)	Get the player object from the number inserted.
+ void addTomato_AMOUNT(int tomato_AMOUNT)	Add the tomato amount with the given number.
+ void addCabbage_AMOUNT(int cabbage_AMOUNT)	Add the cabbage amount with the given number.
+ void addFish_AMOUNT(int fish_AMOUNT)	Add the fish amount with the given number.
+ void addScoreCount(int score)	Add the score with the given number.

+ void addCoinCount(int coin)	Add the money with the given number.
+ GETTER & SETTER of the rest fields	

7.4 class GameMap

This class manages with the map where the player can interact with the other stuffs and contains data about all the blocks in the map.

7.4.1 Field

Name	Description
- int width	Total blocks in horizontal line
- int height	Total blocks in vertical line
- Cell[ ][ ] cellmap	Array containing all cell in a map

7.4.2 Constructor

Name	Description
+ GameMap(String[ ][ ] map)	Processing all the char code in the map array, decode it into a block and set that block into each cell.

7.4.3 Method

Name	Description
+ void printMap()	Print out all the char code in a table form which represent the block in a map.
+ void printEachCell()	Print out all the detail about each cell in a game map.
+ boolean setBlock(Block b, int x, int y)	Assign a block into the cell which located at the coordinate given. Return the result of assigning.
+ boolean isMovePossible(Player p, int targetx, int targety)	Check the map that it is possible that player p can move to the given coordinate. Return the result of checking.

+ boolean isInteractPossible(Player p, int targetx, int targety)	Check that if the player p can interact with the stuff at the given coordinate, Return the result of checking.
+ boolean interactWithBlockTarget(Player p, int targetx, int targety, int interactType)	If it is possible to interact with the stuff located at the given coordinate, return the result of interacting or cooking depends on the interactType number.
+ boolean interactWithAnotherPlayer(Player p, int targetx, int targety)	If there is another player standing on the given coordinate, then return true. Otherwise, return false. (If it is a single-player mode, always return false)
+ void updateIsAnyBlockDownward()	Check that if each block has any block located downward or not, if yes setAnyBlockDownward at this block to true. Otherwise, set to false. (Use for rendering images)
+ void addAllBlocktoRenderableHolder()	Add all the block inside the map to the RenderableHolder. (This method is called when the system completing initializing the map)
+ GETTER for cellmap	
+ GETTER & SETTER for width, height	

7.5 class Sprites

This class contains the char code representing for each block in the game.

7.5.1 Field

Name	Description
+ final char Station	'A'
+ final char TomatoStorage	'B'
+ final char CabbageStorage	'C'
+ final char FishStorage	'D'
+ final char DishPicker	'E'
+ final char Bin	'F'

+ final char Obstacle	‘G’
+ final char FoodCounter	‘H’
+ final char CuttingBoard	‘I’
+ final char FryingPan	‘J’
+ final char Space	‘O’

8. Package meal

8.1 class **FriedFish** extends Menu

8.1.1 Constructor

Name	Description
+ FriedFish(int timeMax)	Initializes the fields for the FriedFish. For the values not given in the constructor, they are fixed as follow: - Name = “Fried Fish” - Price = 50, Max_score = 30 - MenuImage get the image from the RenderableHolder - ingredients is an ArrayList containing 1 fried fish.

8.2 class Menu

This class provide basic fields and methods for all menu.

8.2.1 Field

Name	Description
# String name	The menu’s name
# int price	The money you would gain from serving the menu
# int max_score	The score you would gain from serving the menu
# ArrayList<Ingredient> ingredients	The recipe of the menu List containing all needed ingredient to cook the menu.
# int timeMax	The time limit you have to serve the order.
# Image menuImage	The menu’s image
# double timeLeft	The time left until the menu will disappear from the orders.
# boolean isSend	State that is this menu has been already served yet.

8.2.2 Constructor

Name	Description
+ Menu(int timeMax)	Set isSend to false and initialize the timeMax and timeLeft field with timeMax inserted.

8.2.3 Method

Name	Description
+ boolean isAllIngredients(Entity e)	Check that e is a dish and containing all the ingredient needed to complete the menu or not. Return the result of checking.
+ String toString()	Return String which represents some details about this menu.
+ GETTER & SETTER for each field	

8.3 class OrderManager

This class manage all the incoming order from the customers.

8.3.1 Field

Name	Description
- ArrayList<Menu> orders	List containing all the incoming orders at a moment

8.3.2 Constructor

Name	Description
+ OrderManager()	Initialize the orders field by creating an empty ArrayList.

8.3.3 Method

Name	Description
+ boolean addOrder(Menu m)	If the order number is still not over the limit, add the menu into the orders.
+ Menu removeOrder(int index) throws RemoverOrderFailedException	Remove the order located at the index inserted in the orders field and return the removed order. If the index inserted is the negative number or out of bound of the Array size, throw new RemoveOrderFailedException with the error message.
+ boolean sendOrder(Player p)	- Get the dish being held by the player and check that if the dish is matched with any order in the orders. - If it is matched more than 2 orders, we will select the order that has the least timeLeft. - If there is no order matched with the player's dish, remove the dish and deduct score for 5 points and return false. - Otherwise, remove the matched order from the orders, add coin and money gained from serving order to the GameController, and return true.
+ int typemenu()	This method is use for randoming numbers from 1 to 9 in order to random add menu to the orders.
+ void printTimeLeftOfEachMenu()	Print out the timeleft for all of the menu in orders.
+ void updateOrderNumber()	Checking that is there menu that ran out of time, then remove that menu and deduct score for 5 points per menu.
+ void addMenu()	Randoming add menu to the orders with the number randomed from typemenu().
+ GETTER of orders	

8.4 class Salad extends Menu

8.4.1 Field

Name	Description
# int saladType	Number indicating what salad type is (0 = Simple Salad, 1 = Sashimi Salad)

8.4.2 Constructor

Name	Description
+ Salad (int timeMax, int saladType)	Initializes the fields for the Salad. For the values not given in the constructor, they are depended with the salad type number. - If the salad type is equal to 0 - Name = “Simple Salad” - Price = 50, Max_score = 40 - ingredients is an ArrayList containing 1 sliced cabbage + 1 sliced tomato - If the salad type is equal to 1 - Name = “Sashimi Salad” - Price = 90, Max_score = 60 - ingredients is an ArrayList containing 1 sliced cabbage + 1 sliced tomato + 1 sliced fish - Getting the menu image from the RenderableHolder

8.4.3 Method

Name	Description
+ GETTER and SETTER of saladType field	

9. Package screen

9.1 class **ButtonEndScreen** extends HBox

This class is used to be buttons in the EndScreen class.

9.1.1 Field

Name	Description
+ Button quitButton	The quit button.

9.1.2 Constructor

Name	Description
+ ButtonEndScreen()	- Set space to 100, Set prefWidth to 700. - Set prefHeight 150, Set Alignment to the bottom center, Set padding to 75. - Set the name of the quit button to “Quit”. - Set prefSize of the button to (300,75). - Set font to font size 30. - Set background to LIGHTBLUE.

9.1.3 Method

Name	Description
+ void setupExitButton()	Set quit button that it will exit when I click.
+ void setuptButton()	Set the buttons to change the color when enter the buttons or exit the buttons and other effect.

9.2 class ButtonStartScreen extends VBox

9.2.1 Field

Name	Description
+ Button playButton	The play button.
+ Button quitButton	The quit button.
+ Button p1Button	The 1 player button.
+ Button p2Button	The 2 player button.

9.2.2 Constructor

Name	Description
+ ButtonStartScreen()	- Set space to 100, set prefWidth to 700, set prefHeight to 150 and set Alignment to CENTER RIGHT - Set the name of the play button to “Play” - Set PrefSize to (300,75) - Set font to font size 30 - Set the name of the play button to “Quit” - Set the name of the play button to “1 Player” - Set the name of the play button to “2 Player” - Set every button’s background color to GREEN

9.2.3 Method

Name	Description
+ void Button setupExitButton()	Set up the quit button when click it
+ void Button setUpButton()	Set the buttons to change the color when enter the buttons and exit the buttons and other effect.
+ void Button setPlayerButton()	Set up the play button to show the 1 player and 2 player buttons when click the play button.

9.3 EndScreen

9.3.1 Field

Name	Description
- Stage primaryStage	The main stage of the application.
- Canvas canvas	Canvas for end screen.
- GraphicsContext gc	GraphicsContext for end screen.
- ButtonEndScreen menu	The button from the ButtonEndScreen.
- AnimationTimer endScreenSong	AnimationTimer to play an end screen song.
+ StackPane root	Root for end screen.
- final Font mainfont	Initialize it with the supermarket font and font size 100.
- final Font scorefont	Initialize it with the supermarket font and font size 50.
- final Font detailfont	Initialize it with the supermarket font and font size 40.

9.3.2 Constructor

Name	Description
+ EndScreen(Stage primaryStage)	Set Stage, initialize new canvas with prefSize (1000,800), create new StackPane, set scene, set Title, set Resizable. Call method draw.

9.3.3 Method

Name	Description
+ draw(GraphicsContext gc)	- Set line width to 2, and use mainfont. - Initialize variable s to score. - Initialize variable bonus to money/10. - Initialize variable score to (s+bonus). - Set the specific text with the color depends on the score we get and draw it. - Draw an image to fill in a canvas. - Set the text to show all of the information such as successful dish, and failed dish.

	- Set the animationTimer to cycle the song when it is not running.
+ drawLine(GraphicsContext gc)	- Draw a line between the text.
+ drawScoreandBonus(GraphicsContext s, int s, int bonus)	- Set text and draw it. First one is Score. Second one is Money Bonus. Third one is Total score. - Set the font with scorefont.

9.4 class GameScreen

9.4.1 Field

Name	Description
- Stage primaryStage	The main stage of the application.

- Canvas gameCanvas	Canvas for a game screen.
+ GraphicsContext gamegc	GraphicsContext for game screen.
+ GraphicsContext timegc	GraphicsContext for time screen at the top right.
+ final int draw_origin_x	Initialize it to 48.
+ final int draw_origin_y	Initialize it to 24.
+ final int pixel	Initialize it to 64.
+ int gametime	Time in the game.

9.4.2 Constructor

Name	Description
+ GameScreen(Stage primaryStage, int numberofPlayer)	- Initialize the primaryStage. - Initialize the game via GameController. - Initialize all the pane via SimulationManager - Arrange all the pane and orderly put them on the screen as shown in figure. - Set gametime to max game time - Initialize an Animationtimer that will be a timer if the timer reaches 0, set isTimeUp to true and stop this AnimationTimer. If it has 20 seconds left, play the almost_Time_Up song. - Initialize new AnimationTimer that will run methods that need to be updated all the time. - In this AnimationTimer, the song will play, and load the EndScreen.

9.4.3 Method

Name	Description
- void addListener(Scene s, GraphicsContext gc)	Receive an input from keyboard.
+ void paintGameScreenComponent()	Draw everything in the game screen such as floor, background, all block and entity.

+ void drawTime()	Draw the time with the font size 30 and white color.

9.5 class StartScreen

9.5.1 Field

Name	Description
- Stage primaryStage	The main stage of the application.
- Canvas canvas	Canvas for the screen.
- GraphicsContext gc	GraphicsContext for the screen.
+ StackPane root	StackPane for the screen.
- ButtonStartScreen menu	Button from the ButtonStartScreen.
- AnimationTimer startScreenSong	AnimationTimer that will run a song.
- int timer	Initialize it with 0.

9.5.2 Constructor

Name	Description
+ StartScreen(Stage primaryStage)	- Set the primaryStage. - Create new canvas. - Create ButtonStartScreen. - Call setupButton()

9.5.3 Method

Name	Description
+ void draw(GraphicsContext gc)	- Initialize new StackPane with the variable name root. - Set prefSize of the StackPane. - Set title to “Umm!! Aroiii”. - Set Resizeable to false. - Add canvas in root.

	- Create new AnimationTimer to cycle song when it isn't playing and set the time to wait a second. After that the menu will be added in root.
+ void setBackground()	Draw an image and put it in the canvas and set the color text to black.
+ void setupButton()	- Set on action when click in a playButton, p1Button, and p2Button. - A playButton will setup p1Button and p2Button. - A p1Button will initialize gameScreen in map1. - A p2Button will initialize gameScreen in map2.
+ StackPane getRoot()	Get the root.

10. Package sharedObject

10.1 class AudioLoader

This class stores all the audio and song used in the application.

10.2 interface IRenderable

This interface defines method for drawing Entity into the screen.

10.2.1 Method

Name	Description
+ int getZ()	The Z number is related to the order of rendering image on the screen.
+ void draw(GraphicsContext gc)	Draw the entity image on its current position.
+ boolean isVisible()	State that the entity is still visible or not.

10.3 class RenderableHolder

This interface stores all the image used in the application and manage with drawing all the visible entity on the screen.

10.3.1 Field

Name	Description
- ArrayList<IRenderable> entities	List containing all renderable instance
- Comparator<IRenderable> comparator	Comparator used for sorting the renderable entity in the entities
- final RenderableHolder instance	Singleton of the RenderableHolder class
+ IMAGES	

10.3.2 Constructor

Name	Description

+ RenderableHolder()	Initialize the entities and the comparator.

10.3.3 Method

Name	Description
+ void add(IRenderable entity)	Adding renderable entity into the entities.
+ void update()	Updating all updatable entity in the entities and removing all invisible entity from the entities. This method is being called all the time while the game is running.)
+ void loadResource()	Loading all image with Classloader
+ RenderableHolder getInstance()	Getter method of instance.
+ GETTER for entities	