โครงงาน HeyCream!

ผู้จัดทำ

นายณัฐพงศ์ จันทร์เพ็ง

รหัสนักศึกษา 6604062630188

เสนอ

ผู้ช่วยศาสตราจารย์ สถิตย์ ประสมพันธ์

โครงงานนี้เป็นส่วนหนึ่งของวิชา Object Oriented Programming

ภาคเรียนที่ 1 ปีการศึกษา 2568

สาขาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ประยุกต์

มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญของโครงงาน

โครงงานนี้จัดขึ้นเพื่อวัดผลการเรียนในรายวิชา Object Oriented Programming โดยนำเนื้อหาและความรู้ในบทเรียนมาประยุกต์ เพื่อใช้สร้างผลงานในรูปแบบเกม โดมเกมนี้มีจุดมุ่งหมายในการ ฝึกการจดจำ ความว่องไว และเพื่อความสนุกสนาน

1.2 วัตถุประสงค์ของโครงงาน

1. เพื่อพัฒนาทักษะในการเขียนโปรแกรมเชิงวัตถุในภาษาจาวา
2. เพื่อให้เกิดความเข้าใจในหลักการการเขียนโปรแกรมเชิงวัตถุ
3. เพื่อพัฒนาทักษะในการสร้างเกม
4. เพื่อให้สามารถนำการเขียนโปรแกรมเชิงวัตถุไปประยุกต์ใช้ในงานอื่นๆได้

1.3 ขอบเขตของโครงงาน

HeyCream!

- **รายละเอียดเกม**

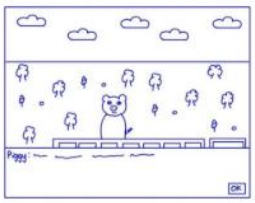
ในช่วงที่เศรษฐกิจตกต่ำคุณจำเป็นต้องหาวิธีสร้างรายได้ให้จงได้! คุณจะได้รับบทบาทเป็นสุนัขที่นำเงินก้อนสุดท้ายมาลงทุนธุรกิจ ice cream food truck โดยคุณจำเป็นต้องรับมือกับเหล่าลูกค้าหลากหลายประเภทให้ได้

- **วิธีเล่น**

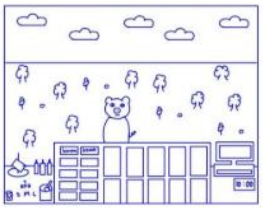
ใช้ mouse ในการควบคุมการเล่นเป็นหลัก เมื่อมีลูกค้ามาจะให้ผู้เล่นทำออเดอร์ตามที่ได้รับ โดยให้เลือกภาชนะสำหรับใส่ไอศครีม ตักลูกไอศกรีมโดยใช้สกู๊ปและตักตามจำนวนที่กำหนด ท็อปปิ้ง ราดซอส โดยจะเล่นจนกว่าเวลาในวันนั้นจะหมด

- ฉาก

เมื่อมีลูกค้ามารอสั่งอาหาร โดยจะมีเวลาระบุว่ากี่โมงแล้ว ตามรูปคือ 10.00 และร้านปิด 22.00 ถือว่าจบเกม



โดยเมื่อลูกค้าสั่ง order จะมี popup ขึ้นมาให้เราอ่านและทำความเข้าใจ



เมื่อกด ok จะนำใบ order ไปเสียบไว้และเราสามารถเริ่มทำได้โดยเรียงจากการเลือกภาชนะ ตักไอศกรีมโดยเริ่มกดที่ scoop ก่อน หลังจากนั้นให้ใส่ท็อปปิ้ง และราดซอสจากนั้นกดให้ลูกค้า

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1. ได้พัฒนาทักษะในการเขียนโปรแกรมเชิงวัตถุในภาษาจาวา
2. ได้เกิดความเข้าใจในหลักการการเขียนโปรแกรมเชิงวัตถุ
3. ได้พัฒนาทักษะในการสร้างเกม

บทที่ 2

ส่วนการพัฒนา

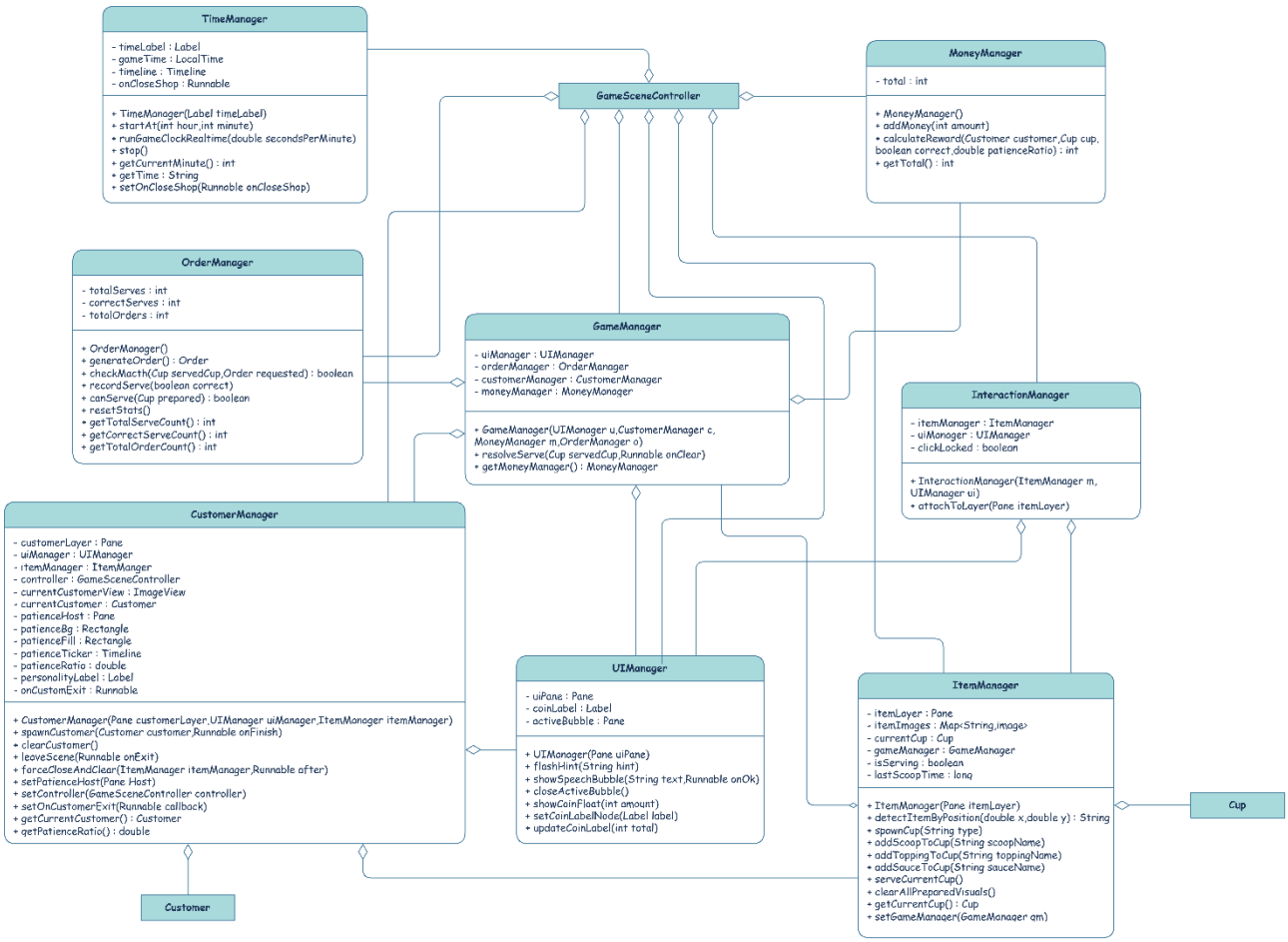
2.1 เนื้อเรื่องย่อ

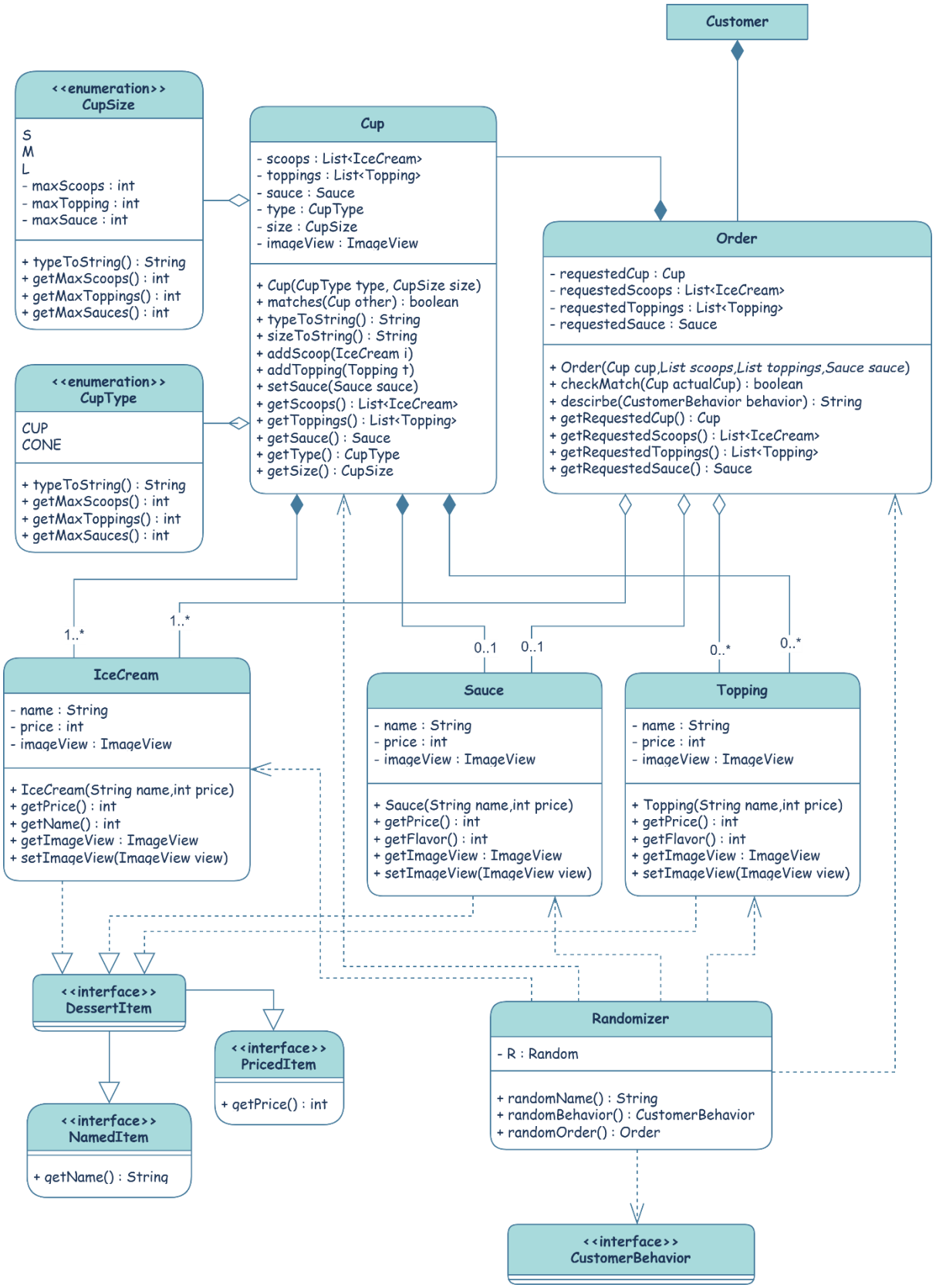
ในช่วงเศรษฐกิจตกต่ำ คุณในฐานะเด็กว่างงานไฟแรงคุณจำเป็นต้องหาวิธีสร้างรายได้ให้จงได้! คุณจะได้รับบทบาทเป็นสุนัขที่นำเงินก้อนสุดท้ายมาลงทุนธุรกิจ Ice cream food truck เพื่อทำให้ธุรกิจของคุณเดินหน้าต่อ คุณจำเป็นต้องรับมือกับเหล่าลูกค้าหลากหลายรูปแบบให้ได้

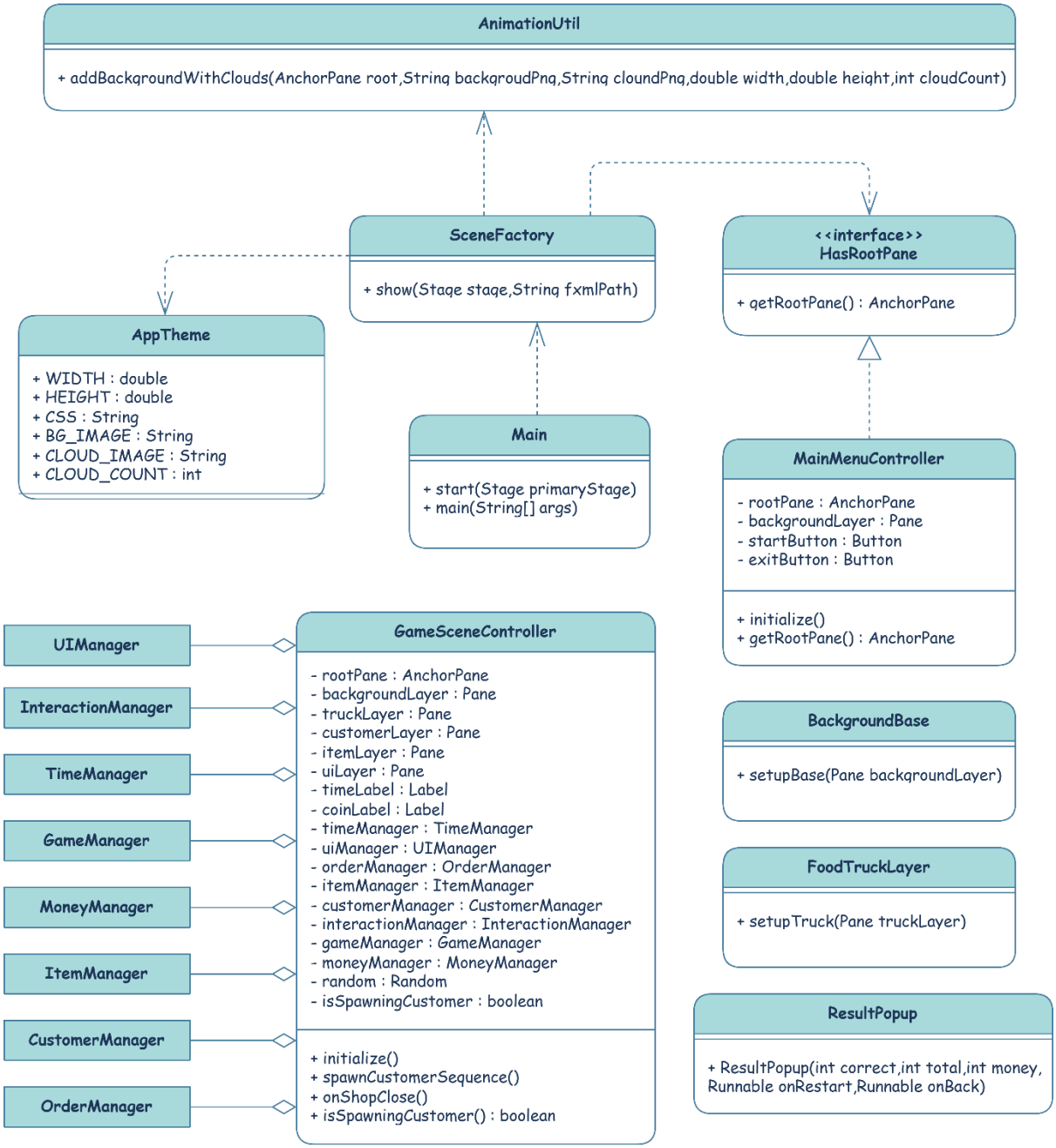
2.2 วิธีการเล่น

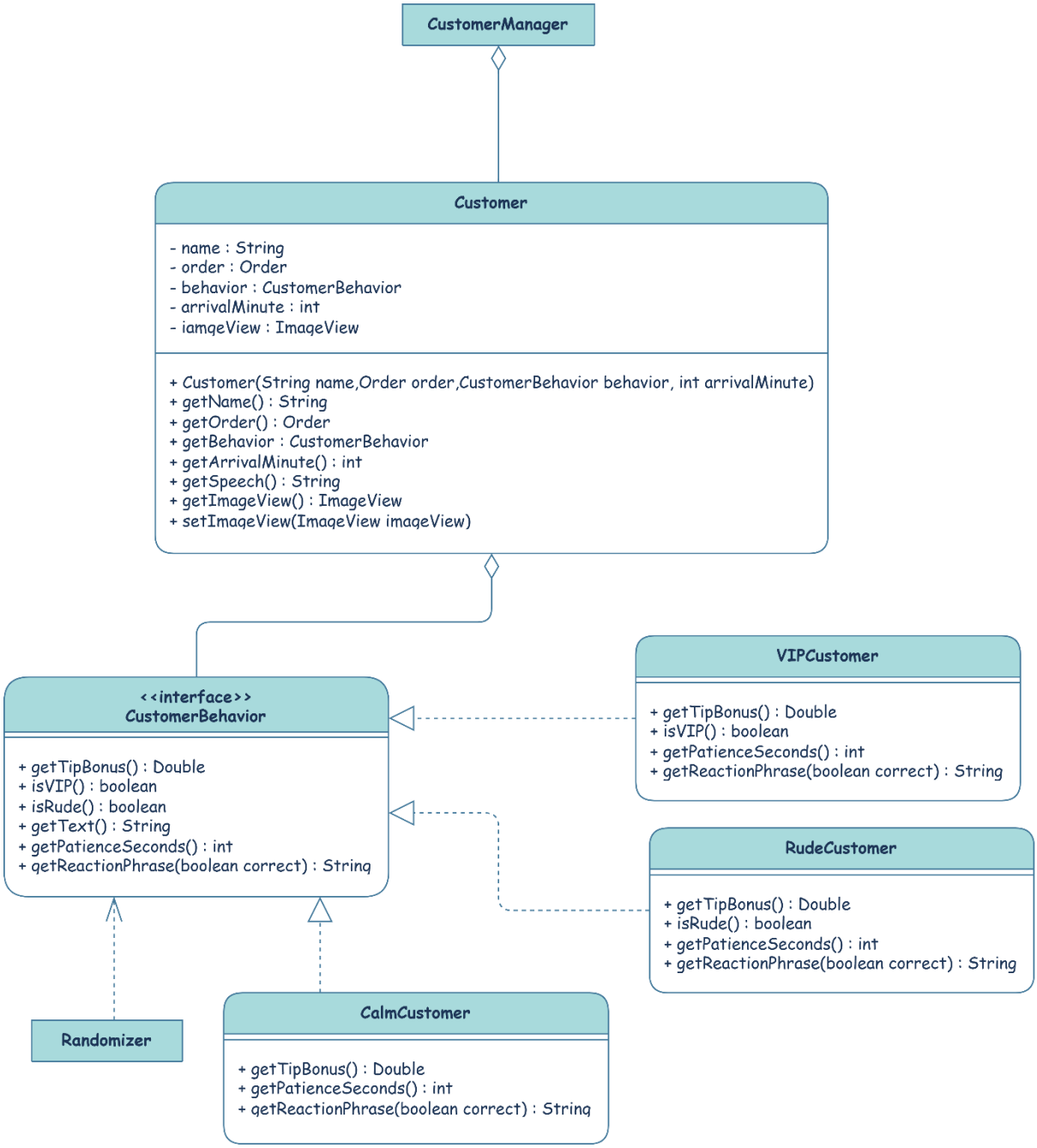
เมื่อเริ่มเกมจะมีลูกค้าเดินเข้ามาเรื่อยๆ โดยลูกค้าแต่ละคนจะมีลักษณะนิสัยที่แตกต่างกันไม่ว่าจะเป็น Calm ,Rude หรือ VIP ซึ่งจะมีความอดทนรอในที่แตกต่างกันออกไป เราจะรับคำสั่งซื้อจากลูกค้าแล้วทำการเตรียมเมนูให้อย่างเป็นลำดับโดยเรียงจาก ประเภทและขนาดของภาชนะ ,รสชาติไอติม ,ท็อปปิ้ง และซอส เมื่อเสร็จให้ทำการเสิร์ฟให้ลูกค้า ซึ่งเราจำเป็นต้องทำให้ทันก่อนที่ลูกค้าจะหมดความอดทน โดยร้านของเราจะเปิดตั้งแต่ 10.00 – 18.00 น.

2.3 Class diagram




Customer
<<enumeration>>
CupSize
S
M
L
- maxScoops : int
- maxTopping : int
- maxSauce : int
+ typeToString() : String
+ getMaxScoops() : int
+ getMaxToppings() : int
+ getMaxSauces() : int
Cup
- scoops : List<IceCream>
- toppings : List<Topping>
- sauce : Sauce
- type : CupType
- size : CupSize
- imageView : ImageView
+ Cup(CupType type, CupSize size)
+ matches(Cup other) : boolean
+ typeToString() : String
+ sizeToString() : String
+ addScoop(IceCream i)
+ addTopping(Topping t)
+ setSauce(Sauce sauce)
+ getScoops() : List<IceCream>
+ getToppings() : List<Topping>
+ getSauce() : Sauce
+ getType() : CupType
+ getSize() : CupSize
Order
- requestedCup : Cup
- requestedScoops : List<IceCream>
- requestedToppings : List<Topping>
- requestedSauce : Sauce
+ Order(Cup cup,List scoops,List toppings,Sauce sauce)
+ checkMatch(Cup actualCup) : boolean
+ describe(CustomerBehavior behavior) : String
+ getRequestedCup() : Cup
+ getRequestedScoops() : List<IceCream>
+ getRequestedToppings() : List<Topping>
+ getRequestedSauce() : Sauce
<<enumeration>>
CupType
CUP
CONE
+ typeToString() : String
+ getMaxScoops() : int
+ getMaxToppings() : int
+ getMaxSauces() : int
1..*
1..*
0..1
0..1
0..*
0..*
IceCream
- name : String
- price : int
- imageView : ImageView
+ IceCream(String name,int price)
+ getPrice() : int
+ getName() : int
+ getImageView : ImageView
+ setImageView(ImageView view)
Sauce
- name : String
- price : int
- imageView : ImageView
+ Sauce(String name,int price)
+ getPrice() : int
+ getFlavor() : int
+ getImageView : ImageView
+ setImageView(ImageView view)
Topping
- name : String
- price : int
- imageView : ImageView
+ Topping(String name,int price)
+ getPrice() : int
+ getFlavor() : int
+ getImageView : ImageView
+ setImageView(ImageView view)
<<interface>>
DessertItem
<<interface>>
PricedItem
+ getPrice() : int
<<interface>>
NamedItem
+ getName() : String
Randomizer
- R : Random
+ randomName() : String
+ randomBehavior() : CustomerBehavior
+ randomOrder() : Order
<<interface>>
CustomerBehavior


AnimationUtil
+ addBackgroundWithClouds(AnchorPane root,String backgroudPng,String cloundPng,double width,double height,int cloudCount)
SceneFactory
+ show(Stage stage,String fxmlPath)
<<interface>>
HasRootPane
+ getRootPane() : AnchorPane
AppTheme
+ WIDTH : double
+ HEIGHT : double
+ CSS : String
+ BG_IMAGE : String
+ CLOUD_IMAGE : String
+ CLOUD_COUNT : int
Main
+ start(Stage primaryStage)
+ main(String[] args)
MainMenuController
- rootPane : AnchorPane
- backgroundLayer : Pane
- startButton : Button
- exitButton : Button
+ initialize()
+ getRootPane() : AnchorPane
UIManager
InteractionManager
TimeManager
GameManager
MoneyManager
ItemManager
CustomerManager
OrderManager
GameSceneController
- rootPane : AnchorPane
- backgroundLayer : Pane
- truckLayer : Pane
- customerLayer : Pane
- itemLayer : Pane
- uiLayer : Pane
- timeLabel : Label
- coinLabel : Label
- timeManager : TimeManager
- uiManager : UIManager
- orderManager : OrderManager
- itemManager : ItemManager
- customerManager : CustomerManager
- interactionManager : InteractionManager
- gameManager : GameManager
- moneyManager : MoneyManager
- random : Random
- isSpawningCustomer : boolean
+ initialize()
+ spawnCustomerSequence()
+ onShopClose()
+ isSpawningCustomer() : boolean
BackgroundBase
+ setupBase(Pane backgroundLayer)
FoodTruckLayer
+ setupTruck(Pane truckLayer)
ResultPopup
+ ResultPopup(int correct,int total,int money, Runnable onRestart,Runnable onBack)


CustomerManager
Customer
- name : String
- order : Order
- behavior : CustomerBehavior
- arrivalMinute : int
- iamgeView : ImageView
+ Customer(String name,Order order,CustomerBehavior behavior, int arrivalMinute)
+ getName() : String
+ getOrder() : Order
+ getBehavior : CustomerBehavior
+ getArrivalMinute() : int
+ getSpeech() : String
+ getImageView() : ImageView
+ setImageView(ImageView imageView)
<<interface>>
CustomerBehavior
+ getTipBonus() : Double
+ isVIP() : boolean
+ isRude() : boolean
+ getText() : String
+ getPatienceSeconds() : int
+ getReactionPhrase(boolean correct) : String
VIPCustomer
+ getTipBonus() : Double
+ isVIP() : boolean
+ getPatienceSeconds() : int
+ getReactionPhrase(boolean correct) : String
RudeCustomer
+ getTipBonus() : Double
+ isRude() : boolean
+ getPatienceSeconds() : int
+ getReactionPhrase(boolean correct) : String
Randomizer
CalmCustomer
+ getTipBonus() : Double
+ getPatienceSeconds() : int
+ getReactionPhrase(boolean correct) : String

2.4 รูปแบบ Application

ใช้ JavaFX application ในการพัฒนาเพราะสามารถตกแต่งได้อย่างสวยงาม และมีการแยกส่วนที่ชัดเจนโดยใช้ FXML สำหรับทำฉาก ,CSS สำหรับกราฟฟิค และ Controller สำหรับตรรกะต่างๆ เป็น Architecture แบบ MVC (Model-View-Controller)

2.5 แนวคิดการเขียนโปรแกรมเชิงวัตถุ (OOP Concept)

2.5.1 ตัวอย่าง Constructor

```
public InteractionManager(ItemManager itemManager, UIManager uiManager)
    {
        this.itemManager = itemManager;
        this.uiManager = uiManager;
    }
```

Constructor InteractionManager เป็น class ที่ทำหน้าที่โต้ตอบเมื่อเรากด item ต่างๆ และเพื่อให้การทำงานของ class นี้อ้างอิงถึง item และ UI เดียวกับที่ GameSceneController ทำงาน จึงใช้การ = เพื่อให้อ้างถึง reference เดียวกัน

```
public Order(Cup requestedCup, List<IceCream> scoops, List<Topping> toppings, Sauce sauce)
    {
        this.requestedCup = requestedCup;
        this.requestedScoops = new ArrayList<>(scoops != null ? scoops : Collections.emptyList());
        this.requestedToppings = new ArrayList<>(toppings != null ? toppings : Collections.emptyList());
        this.requestedSauce = sauce;
    }
```

Constructor Order มีถ้วยจากคำสั่งซื้อ, scoops ที่เป็น List<IceCream> เพื่อเก็บไอติมหลายลูก, toppings ที่เป็น List<Topping> เพื่อเก็บท็อปปิ้งหลายอย่าง และ sauce ที่เก็บเป็นประเภท Sauce

```
public Customer(String name, Order order, CustomerBehavior behavior, int arrivalMinute)
    {
        this.name = name;
        this.order = order;
        this.behavior = behavior;
        this.arrivalMinute = arrivalMinute;
    }
```

Constructor Customer ที่รับ ชื่อลูกค้า (name), คำสั่งซื้อ (order), ลักษณะนิสัย (behavior) และเวลาที่ลูกค้ามาถึง (arrivalMinute) เพื่อใช้คำนวณแถบความอดทน (patience bar)

2.5.2 ตัวอย่าง Encapsulation

```
public ImageView getCurrentCustomerView() { return currentCustomerView; }
public Customer getCurrentCustomer() { return currentCustomer; }
public double getPatienceRatio() { return patienceRatio; }

public void setPatienceHost(Pane host) { this.patienceHost = (host != null ? host : customerLayer); }
public void setController(GameSceneController controller) { this.controller = controller; }
public void setOnCustomerExit(Runnable callback) { this.onCustomerExit = callback; }
```

Encapsulation CustomerManager ในส่วน getter มี getCurrentCustomerView เพื่อคืนค่ารูปภาพ, getCurrentCustomer เพื่อคืนค่าลูกค้าปัจจุบัน, getPatienceRatio เพื่อคืนค่าอัตราส่วนความอดทนของลูกค้าจากหลอดความอดทน และในส่วน setter มี setPatienceHost เพื่อตั้งค่าว่าลูกค้าแต่ละคนมีอัตราส่วนของหลอดความอดทนมากน้อยแค่ไหน, setController ไม่สามารถกำหนดใน constructor เพราะรอการ initialize จึงต้องใช้ setter และ setOnCustomerExit เป็นตัวตั้งสิ่งที่จะเกิดขึ้นเมื่อลูกค้าออกร้าน

```
    public int getTotalServeCount(){ return totalServes; }
    public int getCorrectServeCount(){ return correctServes;
}   public int getTotalOrderCount(){ return totalOrders; }
```

Encapsulation OrderManager ส่วน getter มี getTotalServeCount คืนค่าที่ serve ทั้งหมด, getCorrectServeCount คืนค่าที่ serve ถูก และ getTotalOrderCount คืนค่าจำนวนคำสั่งซื้อ

```
public String getName() { return name; }
public Order getOrder() { return order; }
public CustomerBehavior getBehavior() { return behavior; }
public int getArrivalMinute() { return arrivalMinute; }
public String getSpeech() { return order != null ? order.describe(behavior) : "Hello!";
```

Encapsulation Customer ส่วน getter มี getName คืนค่าชื่อลูกค้า, getOrder คืนค่าคำสั่งซื้อของลูกค้า, getBehavior คืนค่าลักษณะนิสัยของลูกค้า, getArrivalMinute คืนค่าเวลาที่ลูกค้ามาถึง และ getSpeech คืนค่าคำพูดตามลักษณะนิสัย

2.5.3 ตัวอย่าง Composition

```
    private TimeManager timeManager;
    private UIManager uiManager;
    private OrderManager orderManager;
    private ItemManager itemManager;
    private CustomerManager customerManager;
    private InteractionManager interactionManager;
    private GameManager gameManager;
    private MoneyManager moneyManager;
```

GameSceneController เป็น class หลักในการเล่นเกมเลยมีการสร้างตัวแปรที่เป็นตัวจัดการ (manager) ต่างๆ

```
    private final Cup requestedCup;
    private final List<IceCream> requestedScoops;
    private final List<Topping> requestedToppings;
    private final Sauce requestedSauce;
```

Order มีการเก็บ Cup ที่เป็นถ้วยที่ลูกค้าสั่ง, List<IceCream> เก็บไอติมแต่ละรสชาติ, List<Topping> เก็บท็อปปิ้งแต่ละอย่าง และ sauce เก็บซอส

```
    private final UIManager uiManager;
    private final OrderManager orderManager;
    private final CustomerManager customerManager;
    private final MoneyManager moneyManager;
```

GameManager เป็นส่วนที่ใช้ resolveServe จึงต้องติดต่อกับ CustomerManager, OrderManager มี MoneyManager ใช้ add,updateCoin และ UIManager เพื่อแสดงผลข้อความเพิ่มเงินหลัง resolveServe

2.5.4 ตัวอย่าง Polymorphism

```java
public class VIPCustomer implements CustomerBehavior
{
    @Override
    public double getTipBonus() { return 1.5; }

    @Override
    public boolean isVIP() { return true; }

    @Override
    public int getPatienceSeconds() { return 15; }

    @Override
    public String getReactionPhrase(boolean correct)
    {
        return correct ? "Marvelous! You never disappoint!" : "Oh no… that’s not my order.";
    }

}
```

VIPCustomer มีการใช้ @Override ใน method ที่ได้ implement มา

```java
public class Topping implements DessertItem
{
    @Override public String getName() { return name;
}   @Override public int getPrice() { return price; }
}
```

Topping มีการใช้ @Override ใน method ที่ได้ implement มา

2.5.5 ตัวอย่าง Abstract

```java
public class Main extends Application
{
    @Override
    public void start(Stage primaryStage)
    {
        primaryStage.setTitle("HeyCream 🍦");
        primaryStage.getIcons().add(new Image(
            getClass().getResource("/com/heycream/assets/Logo.png").toExternalForm()
        ));
        SceneFactory.show(primaryStage, "/com/heycream/gui/fxml/main_menu.fxml");
    }
    public static void main(String[] args)
    {
        launch(args);
    }
}
```

Main extends Application และมีการ override method start จาก Application

```java
public class ResultPopup extends Pane
{

    public ResultPopup(int correct, int total, int money, Runnable onRestart, Runnable onBack) { ... }
    private int computeStars(int money) { ... }
    private String getStarsSymbol(int stars) { ...}
}
```

ResultPopup extends Pane และมีการ method setPrefSize, getStyleClass, setLayoutX,Y

2.5.6 ตัวอย่าง Inheritance

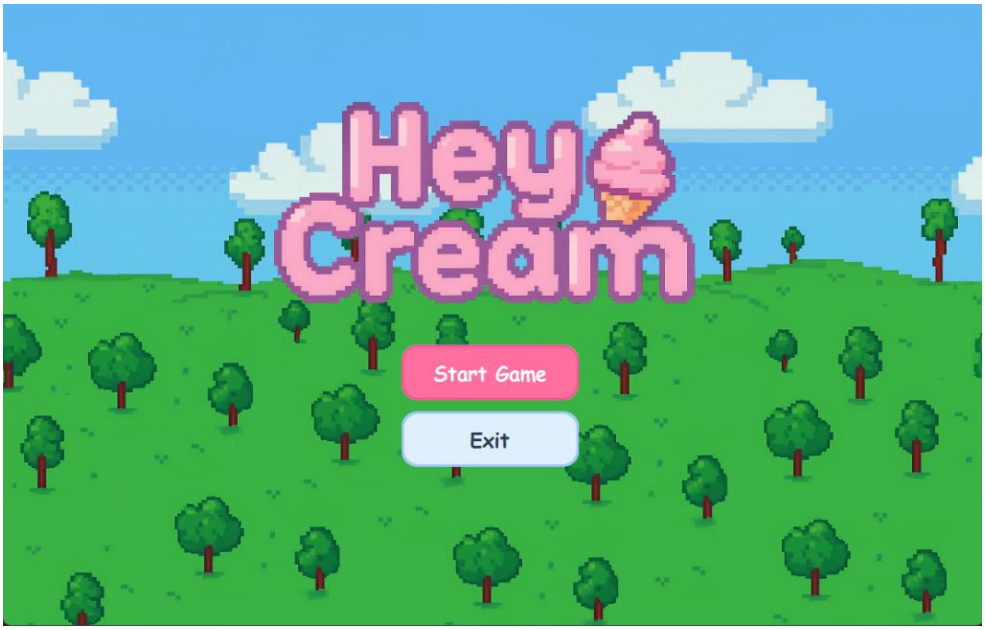
```java
public interface DessertItem extends NamedItem, PricedItem {}
```

DessertItem เป็น interface ที่มีการ extends interface 2 ตัวคือ NamedItem, PricedItem

```java
public class RudeCustomer implements CustomerBehavior
{
    @Override
    public double getTipBonus() { return 0.8; }

    @Override
    public boolean isRude() { return true; }

    @Override
    public int getPatienceSeconds() { return 20; }

     @Override
    public String getReactionPhrase(boolean correct)
    {
        return correct ? "Finally, about time!" : "Ugh, this is terrible!";
    }
}
```

RudeCustomer implements CustomerBehavior เลยต้อง Override method จาก interface

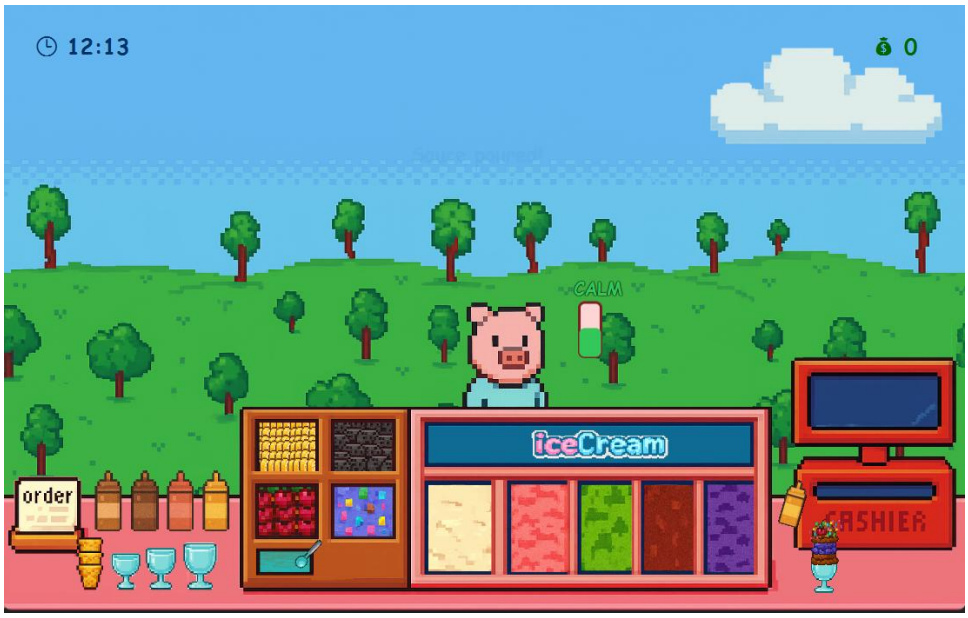
2.6 GUI



หน้า Main menu คลิก Start Game เพื่อเริ่มเล่น

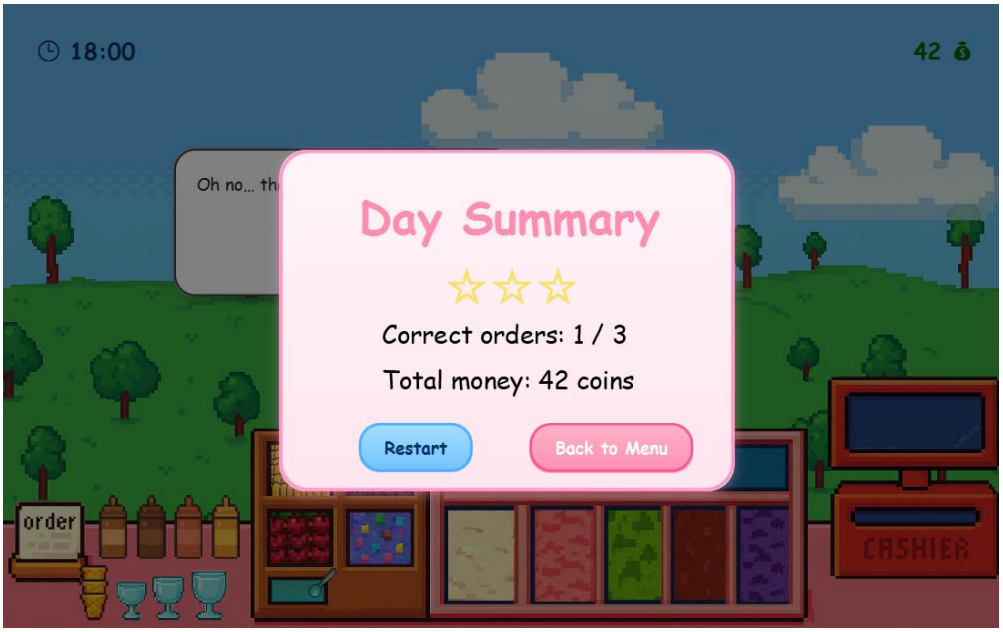
เมื่อเริ่มเกมจะมีลูกค้าเดินมาสั่งเมนู โดยลูกค้าแต่ละคนจะมีลักษณะนิสัยที่แตกต่างกันซึ่งจะมีผลต่อหลอดความอดทนของลูกค้า และ bonusTip จากลูกค้า

เมื่อหลอดความอดทนของลูกค้าหมดจะทำให้สูญเสียลูกค้าไป



โดยหลังจากรับคำสั่งซื้อแล้วให้เราเตรียมเมนูเสิร์ฟ โดยต้องทำให้ตรงและไวเพื่อได้รับเงินที่มากขึ้น

เมื่อเสิร์ฟถูกต้องลูกค้าจะพูดตอบรับกับเมนู และหลังจากนั้นเราจะได้รับเงิน



เมื่อจบเกม (เวลา 18:00) จะทำการแสดงหน้าสรุปผลโดยดาวที่ได้รับจะคำนวณจากจำนวนเงินที่ทำได้

2.7 Event Handling

```
@FXML private AnchorPane rootPane;
@FXML private Pane backgroundLayer;
@FXML private Pane truckLayer;
@FXML private Pane customerLayer;
@FXML private Pane itemLayer;
@FXML private Pane uiLayer;
@FXML private Label timeLabel;
@FXML private Label coinLabel;
```

มีการใช้ FXML เป็นตัวเชื่อมระหว่าง UI กับ controller

```
@FXML
    public void initialize()
    {
        ...
    }
```

Initialize เป็น event lifecycle ซึ่งถือเป็น FXML initialization event handler

```
startButton.setOnAction(e ->
    SceneFactory.show((Stage) startButton.getScene().getWindow(),
        "/com/heycream/gui/fxml/game_scene.fxml")
);
```

ปุ่ม startButton เมื่อถูกกดจะเรียก SceneFactory.show()

```
pour.setOnFinished(e -> itemLayer.getChildren().remove(bottle));
```

เมื่อทำการราดซอสเสร็จ จะทำการ remove ขวดซอสจาก itemLayer

```
exit.setOnFinished(e ->
{
    customerLayer.getChildren().remove(currentCustomerView);
    currentCustomerView = null;
    currentCustomer = null;
    if (onExit != null) onExit.run();
    if (onCustomerExit != null) onCustomerExit.run();
 });
```

เมื่อลูกค้าเดินออกจากฉาก จะลบภาพลูกค้าและเซ็ททุกอย่างเป็น null แล้ว onExit.run() คือการ run คำสั่งที่ส่งมาจากภายนอก ส่วน OnCustomerExit.run() คือหลังจากที่ลูกค้าเดินออกไปทำการ run callback

2.8 Algorithm สำคัญในโปรแกรม

```java
public static String randomName()
    {
        String[] names = {"Tiger", "Elephant", "Pig", "Cat", "Dog"};
        return names[R.nextInt(names.length)];
    }
    public static CustomerBehavior randomBehavior()
    {
        double roll = R.nextDouble();
        if (roll < 0.6) return new CalmCustomer();
        if (roll < 0.85) return new RudeCustomer();
        return new VIPCustomer();
    }

    public static Order randomOrder()
    {
        CupType type = R.nextDouble() < 0.7 ? CupType.CUP : CupType.CONE;
        CupSize size = (type == CupType.CUP)
                ? CupSize.values()[R.nextInt(CupSize.values().length)]
                : CupSize.M;
        Cup cup = new Cup(type, size);
        int maxScoops, maxToppings, maxSauces;
        if (type == CupType.CONE)
        {
            maxScoops = type.getMaxScoops();
            maxToppings = type.getMaxToppings();
            maxSauces = type.getMaxSauces();
        }
        else
        {
            maxScoops = size.getMaxScoops();
            maxToppings = size.getMaxToppings();
            maxSauces = size.getMaxSauces();
        }
        int scoopCount = maxScoops;
        List<IceCream> scoops = new ArrayList<>();
        for (int i = 0; i < scoopCount; i++)
        {
            scoops.add(randomFlavor());
        }
        List<Topping> toppings = new ArrayList<>();
        int toppingCount = R.nextInt(maxToppings + 1);
        for (int i = 0; i < toppingCount; i++)
        {
            toppings.add(randomTopping());
        }
        Sauce sauce = null;
        if (R.nextDouble() < 0.5 && maxSauces > 0)
        {
            sauce = randomSauce();
        }
        return new Order(cup, scoops, toppings, sauce);
    }

    private static IceCream randomFlavor()
    {
        String[] flavors = {"Vanilla", "Strawberry", "Matcha", "Chocolate", "Blueberry"};
        String f = flavors[R.nextInt(flavors.length)];
        return new IceCream(f, 20);
    }

    private static Topping randomTopping()
    {
        String[] toppings = {"Cherrie", "Oreo", "Banana", "Candy",};
        String t = toppings[R.nextInt(toppings.length)];
        return new Topping(t, 10);
    }

    private static Sauce randomSauce()
    {
        String[] sauces = {"Caramel", "Chocolate", "Strawberry", "Honey"};
        String s = sauces[R.nextInt(sauces.length)];
        return new Sauce(s, 8);
    }
```

มีการใช้ random ในการสุ่ม customer, order, ice cream, Topping และ Sauce ทำให้มีการ generate อุปสรรคที่หลากหลายและสร้างสรรค์

บทที่ 3

สรุปและประเมินผล

3.1 ปัญหาที่พบระหว่างการพัฒนา

- มีการใช้ JavaFX ซึ่งไม่เคยเรียนรู้มาก่อนทำให้ในระหว่างพัฒนาต้องคอยเรียนรู้สิ่งใหม่ๆอยู่เสมอ
- มีคลาสที่เยอะมาก ทำให้ในช่วงพัฒนามีบาง attributes หรือ method ที่ไม่ถูกเรียกใช้และต้องลบ
- ตอนแรกใช้การกดแล้วเอามาวางแต่บั๊กเยอะมาก ทำให้ต้องปรับเป็นการกดแล้ว spawn item ขึ้น
- JavaFX ใช้ Scene Builder ได้แต่ไม่รู้

3.2 จุดเด่นของโปรแกรมที่ไม่เหมือนใคร

- ตัวละคร (ลูกค้า) แต่ละคนมีลักษณะนิสัยเฉพาะตัวทำให้เราต้องรู้จักปรับตัวกับความอดทนของแต่ละตัวละคร
- อุปสรรค (คำสั่งซื้อ) จะเป็นแบบสุ่มทำให้เกมมีความหลากหลายและมีความสร้างสรรค์มากต่างจากเกมอื่นๆ
- รางวัล (เงิน) ในแต่ละคำสั่งซื้อมีการคำนวณที่ไม่เหมือนใครโดยมาจาก ราคาพื้นฐานของเมนูบวกกับลักษณะนิสัยเฉพาะตัว และความอดทนของตัวละคร (ลูกค้า)

3.3 สรุปผลการพัฒนา

ผลลัพธ์ออกมาได้เป็นที่น่าพอใจ ถึงจะไม่ตรงตาม proposal ในบางจุดแต่ทำให้เราได้พัฒนาและได้ลองทำอะไรใหม่ๆมากขึ้น