## 

## **Project Report: Breaking 2**

**Done by: Yousef Mureb: b00088535**

**Abdelrahman Ahmed: b00087338**

**Fares sowan: b00085400**

**Omar Alhamayda: b00089085**

## 

## 

## **Overview of the Game**

## Provide a clear description of how the game works. Justify why you think it is a good game and why people would love to play it. Please include the YouTube video link in this section showing how the game works.

Youtube video link: <https://youtu.be/iuoc0mE9ydU>

This is a story-based game that is a parody of Breaking Bad. The player is Walter White, a renowned cookie baker who had a falling out with his partner, Gustavo Fring. Due to this falling out, White decided to leave his and Fring’s joint cookie chain: LosCookiesCabrons. After sometime without baking, Walter realizes that he has nothing but that, so he and Jesse Pinkman (his sous chef) decide to take back their seat in the cookie-baking world. They start by baking in an RV- far from any unwanted eyes to keep Walter’s secret recipe protected.

Walter and Jesse approach their (previous) competitor- Tuco Salamanca - as a possible investor and distributor. A test of sorts soon followed, but with Walter’s quick thinking, he and Jesse get out safe and sound. Walter and Tuco decide to conduct business with each other, due to their new-found respect for each other. Hector, an old friend, approached Walter regarding Gustavo and asks him to give Gustavo another chance.

Walter then goes to meet Gustavo- they either fight or decide to join forces yet again (Tuco doesnt take this too well).

We think this is a good game because it discovers an alternate universe where people take cookies seriously, especially walter white. We also think its good because we try to emulate how we remember the characters acted. We are also proud of the final fight- even if it may be somewhat basic.

## **Description of Game Components**

Provide a description of the components of the game. Each subsection should describe in detail each component of the game.

## **Objective**

Restore Walter’s status as the cookie king, after forfeiting his title due to a run-in with his previous partner, Gustavo Fring. The game allows you to undergo walters story from starting in the RVtrailer to beating Gustavo (or joining him). The game offers choices in certain part that later affect the ending. The player progresses in the game by completing the inputs required which can either be from the sensor or from the console. When reaching the end, the player must face off the enemy in a turn based combat. The player also gets to use a different cookie type based pn the decisions taken in the game.

## **Scenes**

The game includes 3 scenes in the form of levels. These scenes are the RVtrailer, Tucosoffice and LosCookiesCabron. In each scene, character dialogue takes place as well as sensors are used for input. We use the users input for decisions in the game that latter affect the last scene.

The first scene involves a choice of how walter responds to jesse- this will affect the type of cookie made later, as well as walter’s state/mood. Different cookies have different effects, and walter’s mood will have an effect on a choice in the last scene. Then, Walter (the player) bakes a cookie- the cookie baking process involves different sensors, one for each step of the baking process. The player then eats the cookie.

The second scene takes place in Tuco’s office, the player and Jesse meet Tuco. They are there to conduct business with him. Tuco asks Walter why he would want to work with him- Walter’s answer here also affects his state/mood. Tuco then proceeds to taste his cookies, while Walter and Jesse talk amongst themselves. After eating at 3 of any type of cookie, Tuco will have a unique dialogue. All dialogues end up with Tuco wanting to take all the profits for himself, Walter is forced to use a noise-activated flour bomb to escape the situation. After this, they both have a new-found respect for each other and conduct business together. Tuco sets up a meeting later between Walter and Hector, wherein Hector tells Walter to go see Gustavo.

The final scene is in LosCookiesCabron. Gustavo apologizes to Walter, depending on Walter’s mood, the player may decide to forgive him or not (otherwise, Walter’s mood decides for the player). A fight breaks out, wherein a sensor is used to punch.

## **Objects**

Most of the objects used are for sensor purposes. These include hand,whisk, bowl,flour bomb,knife and pan. Each object uses a specific sensor to read the input from the phone to carry out a specific task. Another object we use is the cookie, this is the centerpiece of the game.

## **Characters**

The main 3 characters used are Walter White, Gustavo and Tuco. The player that controls the sensors is essentially Walter, and he carries out all sensor operations. The other two characters are mainly involved in the 2 later scenes. In the last scene, Tuco can fight Walter and he would be known as an enemy. An enemy gets to use 3 different moves like attack, charge attack and block. Each character also gets his own health status bar and base damage dealt. The game involves different ways to increase this and it is demonstrated in the last scene. The character walter also has moods which later affect the game's outcome.(this uses a state design pattern).

## **Sensors**

In the game, 4 main sensors were used. These sensors were the accelerometer sensor,the gyro meter sensor, the average power sensor, and the heading sensor. All sensors used are from the app called sensor log using an Iphone. The sensors were used with each object to model a specific action to take place. Most of the sensors were used in the BakeCookies class, can check it for reference. The sensors are used as an input from the player (Walter white) and each sensor carries out a different output. Sensors are mainly used to bake the cookies or used to attack an enemy.

## 

## **Using Design Patterns**

Briefly explain how each design pattern was used to put together the game. For each design pattern explain where and how it was utilized in your code. Provide code segments (not full code) to explain how each design pattern was implemented.

## **Observer/Singleton**

We used the sinegelton design pattern for a couple classes, naimly walter,UI,Watch and ControlPanel since we only need one of each instance. in TucosOffice, we used the observer design patter. The Watch used is a subject which publishes a message to tucosOffice. The watch will wait for 5 seconds waiting on the user to apply an input using a sensor, if the user doesnt not use the sensor in time, the game ends.

**public** **static** **synchronized** Watch getInstance() {

**if** (*instance*==**null**) *instance*=**new** Watch();

**return** *instance*;

}

**public** **void** start() {

start=**true**;

}

**public** **void** stop() {

end=**true**;

}

**public** **void** run() {

**while**(!start){

**try** {

t.*sleep*(10);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**while**(secondsPassed<4 && !end) {

**try** {

TimeUnit.***SECONDS***.sleep(2); //is this right?

secondsPassed++;

}**catch**(InterruptedException e) {e.printStackTrace();}

**if**(secondsPassed==4 && !end)

UI.*print*((5-secondsPassed) + " SECOND LEFT!!! HURRY");

**else** **if**(!end)

UI.*print*((5-secondsPassed)+ " seconds left!!");

}

**if**(!end) {

UI.*print*("it is the end of the time! ");

Message m=**new** Message(**this**, "watch", secondsPassed+ " seconds have passed");

publishMessage(m); //notifies player that GAMES OVER

}

}

We also used the observer design pattern using the Tuco object. Tuco will randomly eat a cookie and depending on the cookie type, will publisb a message to tucos office, while that is happening, a dialogue between jesse and walter takes place.

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

Random rand = **new** Random();

**int** choice;

**while**(sleepyStatus!=3 && spicyStatus!=3 && normalStatus!=3)

{

choice = rand.nextInt(3);

**if**(choice==0)

{

UI.*print*("\nTuco(muttering): hmmmm spicy- who makes spicy cookies?\n");

spicyStatus++;

}

**else** **if**(choice==1){

UI.*print*("\nTuco(muttering): this one got me feelin a lil dizzy\n");

sleepyStatus++;

}

**else** **if**(choice==2){

UI.*print*("\nTuco(muttering): tasty \n");

normalStatus++;

}

sleep(2\*longSleep);

}

Message m= **new** Message(**this**,"Tuco","done");

publishMessage(m);

UI.*print*("Tuco: Enough with you two!");

sleep(shortSleep);

**if**(sleepyStatus==3){

UI.*print*("Tuco:... mannn");

sleep(longSleep);

UI.*print*("Tuco: You guys.. make some GOOD cookies");

sleep(longSleep);

UI.*print*("Tuco: but MaN....");

sleep(longSleep);

UI.*print*("Tuco: soMethIng's gottA Be up wiTh ThEEse..");

sleep(longSleep);

UI.*print*("he takes a sip of something");

sleep(shortSleep);

UI.*print*("Tuco: and you know, I can't help but think...");

sleep(shortSleep);

UI.*print*("Tuco: that you're tryna do somethin to me y'know?");

UI.*print*("Tuco: and I dont appreciate that");

m=**new** Message(**this**, "Tuco", "sleepy");

}

**else** **if** (spicyStatus==3)

{

UI.*print*("Tuco: What did you guys put in this?? Why is it so spicy!! ");

sleep(shortSleep);

UI.*print*("Tuco: I mean, these are cookies man! They're supposed to be sweet, make you feel better!");

sleep(shortSleep);

UI.*print*("Tuco: But this.. this just burns me up");

m=**new** Message(**this**, "Tuco" , "spicy");

}

**else** **if** (normalStatus==3)

{

UI.*print*("Tuco: those were some of the best cookies i have ever had");

sleep(shortSleep);

UI.*print*("Tuco: you guys make an excellent team and i love the work you put out");

sleep(shortSleep);

UI.*print*("Tuco: With that said, I would like Jesse to work for me");

sleep(shortSleep);

UI.*print*("i will pay you well Jesse and personally gift you a brand new car");

sleep(shortSleep);

UI.*print*("Tuco: So, what do you say? ");

m=**new** Message(**this**, "Tuco", "normal");

}

publishMessage(m);

}

**else** **if**(num==1){

UI.*print*("Walter (whispering): He wants to taste MY cookies? Who does he think he is questioning the cookie king..");

sleep(shortSleep);

UI.*print*("Jesse (whispering): Well, he's a professional, he's gotta insepct tha goods don't he?");

sleep(longSleep);

UI.*print*("Jesse (whispering): I mean besides.... both of yall might be surprised..");

sleep(shortSleep);

UI.*print*("Walter (whispering): What do you mean surprised? I made a perfect batch just for this occasion");

sleep(shortSleep);

UI.*print*("Walter (whispering): Jesse, what batch of cookies did you get??");

sleep(shortSleep);

UI.*print*("Jesse (whispering): Mista White i got the ones on the brown counter");

sleep(shortSleep);

UI.*print*("Walter (whispering):YOU DID WHAT????");

sleep(shortSleep);

UI.*print*("Walter (whispering): Thats the batch with the diffirent cookies from when you tried to mess with me");

sleep(shortSleep);

UI.*print*("Jesse (whispering): I didn't know you didnt even mention it mann");

sleep(longSleep);

**if**(!end)

UI.*print*("Jesse (whisepring): he sure is takin his time though");

**while**(!end){

sleep(10);

}

}

## **Template Method**

## The template method was implemented in the BakeCookies class, this is also the class with the most sensors used. Seeing as template method was described as a recipe, we thought it would make sense to use it in a recipe:

void prepareCookies() {

chopChocolate();

mixTogether();

addIngredients();

placeInOven();

}

The following is our implementation of a template method, this was used in BakeCookies.

## **Strategy**

## Strategy design pattern was mainly implemented in cookies, as “CookieBehaviour”. CookieBehaviour can either be spicy, normal or sleepy.

public interface CookieBehaviour {

public void eat() throws InterruptedException;

public String getName();

}

public class NormalBehaviour implements CookieBehaviour {

String name="normal";

public String getName(){

return name;

}

public void eat(){

UI.*print*("Walter: Ah, I truly haven't lost my touch- another chemically sound cookie");

}

}

## **State**

## The state design pattern was mainly used with the player Walter White. Walt has 3 states, which are neutral, friendly and aggressive. He starts out as neutral and depending on the choices the user takes throughout the game, the states transition. The states have an importance in the last scene, where depending on the state, the player can either team with Gustavo and face Tuco, or face Gustavo alone. We also implemented the states in the scene design where at the end of the each scene, the level transitions from one to another. Here are walters states:

**public** **class** AggressiveState **implements** WaltState {

String name="aggressive";

**public** **void** next(Walter walt, String nextString) {

// **TODO** Auto-generated method stub

**if**(nextString.toLowerCase().contains("neutral"))

walt.setState(**new** NeutralState());

**else** **if**(nextString.toLowerCase().contains("friendly"))

walt.setState(**new** FriendlyState());

// else if(nextString.toLowerCase().contains("aggresive"))

// UI.print("");

}

@Override

**public** **void** printStatus() {

// **TODO** Auto-generated method

}

**public** String getName(){

**return** name;

}

}

**public** **class** FriendlyState **implements** WaltState {

String name="friendly";

@Override

**public** **void** next(Walter walter, String state) {

// **TODO** Auto-generated method stub

**if**(state.toLowerCase().contains("aggressive"))

walter.setState(**new** AggressiveState());

**else** **if**(state.toLowerCase().contains("neutral"))

walter.setState(**new** NeutralState());

// else if(state.toLowerCase().contains("friendly"))

// /UI.print("walt is friendly");

}

@Override

**public** **void** printStatus() {

// **TODO** Auto-generated method stub

UI.*print*("walt is now friendly");

}

**public** String getName(){

**return** name;

}

}

**public** **class** NeutralState **implements** WaltState {

String name="neutral";

@Override

**public** **void** next(Walter walter, String state) {

// **TODO** Auto-generated method stub

**if**(state.toLowerCase().contains("aggressive"))

walter.setState(**new** AggressiveState());

// else if(state.toLowerCase().contains("neutral"))

// //ntn

**else** **if**(state.toLowerCase().contains("friendly"))

walter.setState(**new** FriendlyState());

}

@Override

**public** **void** printStatus() {

// **TODO** Auto-generated method stub

UI.*print*("Walt is neutral now ");

}

**public** String getName(){

**return** name;

}

}

## **Command**

Command Design pattern was used to perform the actions/responses of the choices that the player was presented with. For example, the first choice presented to the player was Walter’s response to Jesse. There were three possible responses, each with their own class and effects. A control panel was used to execute the right response, depending on the user’s input.

A response class:  
public class WalterResponse1 implements Command {

Walter walt;

public WalterResponse1()

{

walt= Walter.*getInstance*();

}

*@Override*

public void execute(Cookie cookie) {

// **TODO** Auto-generated method stub

walt.response1(cookie);

}

}

The Command Interface:

public interface Command {

void execute(Cookie cookie);

}

The Control Panel:

import java.util.ArrayList;

public class ControlPanel {

static ArrayList<Command> *slots*;

private static ControlPanel *instance*;

private ControlPanel(ArrayList<Command> slots) {

this.*slots*=new ArrayList<Command>();

for(int i=0; i<slots.size(); i++){

this.*slots*.add(slots.get(i));

}

}

public static synchronized void add(Command slot){

*slots*.add(slot);

}

public static synchronized ControlPanel getInstance(ArrayList<Command> slots){

if(*instance* == null){

*instance* = new ControlPanel(slots);

}

return *instance*;

}

public static synchronized ControlPanel getInstance(){

if(*instance* == null){

return null;

}

return *instance*;

}

public void buttonWasPressed(int index, Cookie cookie){

*slots*.get(index).execute(cookie);

}

}

## 

## 

## **Using Multithreading**

Explain how and why multi-threading was used in the game. Clearly explain each thread and how synchronization was carried out.

Multithreading was used though the watch class aswell as tucos office.in the game, tuco eats different kinds of cookies and based on the cookie type he reacts a certain way. While this is happening, jesse and walter have a conversation together. Moreover, a watch was used that ticks as time progresses while it waits for the user to insert an input. Refer to the code in the observer/singelton section for a reference.

## **Using Sensors**

Explain how and which sensors were used in the game. Clearly explain each sensor and the algorithms used to process it.

As previously mentioned, we used four sensors in this game (some sensors were used multiple times, for the sake of conciseness, we will only talk about one implementation of each): accelerometer, gyrometer, average power (sound) and heading sensors.

*Accelerometer:*

The accelerometer was used to implement the punching motion- we took the magnitude of the x,y,z accelerations to get the magnitude of our acceleration. We thought of this as the force of the punch.

String Zvalue = (String) jsonObject.get("accelerometerAccelerationZ");

String Xvalue = (String) jsonObject.get("accelerometerAccelerationX");

String Yvalue = (String) jsonObject.get("accelerometerAccelerationY");

double accZ = Double.*parseDouble*(Zvalue);

double accY= Double.*parseDouble*(Yvalue);

double accX = Double.*parseDouble*(Xvalue);

aggMag= Math.*sqrt*((accZ\*accZ)+(accX\*accX)+(accY\*accY));

if(aggMag>2.5 && aggMag<=10.0 ){

setPunch(true);

}

*Gyrometer*:

The gyrometer was used when implementing the whisk object. We made the user model using a whisk using there phone in order to combine the cookie batter together. This was necessary in the whole process of creating the cookies which used the template design pattern. We noted down the rotation rate of the X,Y and Z plane and from there decided on some parameters.

String Zgyro = (String) jsonObject.get("gyroRotationZ");

String Xgyro = (String) jsonObject.get("gyroRotationX");

String Ygyro = (String) jsonObject.get("gyroRotationY");

**double** gyroX= Double.*parseDouble*(Xgyro);

**double** gyroY= Double.*parseDouble*(Ygyro);

**double** gyroZ= Double.*parseDouble*(Zgyro);

**if**(((gyroZ>=2 && gyroZ<=5) || (gyroZ<=-3 && gyroZ>=-7)) && ((gyroX>=4 && gyroX<=8) || (gyroX<=-4 && gyroX>=-8)) && ((gyroY>=3 && gyroY<=7) || (gyroY<=-2 && gyroZ>=-5))){

//UI.print("Whisk is used! ");

setMix(**true**);

}

*Average Power (audio/sound):*

The average power sensor was used to implement the sound-activated flour bomb in Tuco’s office. Once the user made a sound that exceeded a given threshold, we considered that as an activation of the bomb, making use of it.

tring soundPeak = (String) jsonObject.get("avAudioRecorderPeakPower");

double peakSound= Double.*parseDouble*(soundPeak);

if(peakSound>=-0.9){

//UI.print("WEEE");

setClap(true);

}

*Heading*:

The heading sensor was used to model the pouring the batter from the bowl to the tray. The sensor parameter uses was locationTrueHeading and was mainly used to determine a rotation with respect to height.

String locHeading = (String) jsonObject.get("locationTrueHeading");

**double** heading= Double.*parseDouble*(locHeading);

**if**(heading<=100 || heading>=300){

setPour(**true**);

}

Progress (include whatever is done ):

| Requirement | Status |
| --- | --- |
| Having a complete Hierarchy | To be completed |
| Implementing Observer Design Pattern | Done |
| Implementing Singleton Design pattern | Done |
| Implementing Command Design Pattern | Done |
| Implementing State Design Pattern | Done |
| Implementing Template Design Pattern | Done |
| Implementing Strategy Design Pattern | Done |
| 3 or more Locations | Done |
| 3 or more characters | Done |
| Implementing 3 or more sensors | completed |
| Having Sensors deeply embedded in the game |  |
| Writing the story class | To be completed |
| Implementing Message based architecture | completed |
| Implementing Multithreading | completed |
| Finalizing the Report | To be completed |