Bilkent University

Department of Computer Science

Senior Design Project

KEBAP TYCOON

Low-Level Design Report

Group Members

Can Akgün
Doğancan Demirtaş
Kıvanç Kamay
Sinem Sav
Ulaş Sert

Supervisor

Hakan Ferhatosmanoğlu

Jury Members

Uğur Güdükbay
Selim Aksoy

Innovation Expert
Armağan Yavuz

Table Of Contents

1. Introduction

One of the contemporary sectors included in computer science field is surely entertainment sector. Today, the game industry has a big portion in economics; it employs dozens of people for different positions starting from game designer, game developer and all the way to the marketing executive. In this big market, business simulation games which are known as tycoon games became one of the most popular genres.

Businesses usually have tight set of rules and revolve around scarce resource allocation. Thus, in a way, maintaining a business in real world already is a sort of game. Consequently, if one wants to play that game, it’s actually much easier to just turn on an electronic device and play one of the many great business tycoon simulations.

Within this genre, we decided to build a Turkish food business game, called Kebap Tycoon. This game will represent humorous sides of Turkish food and restaurant culture in which Turkish people can adore as well as where players can found their restaurant chains and put effort to be the best in this sector by making decisions on both preferences of sales and locality strategies.

With this report, component level of design process of our senior design project will be explained so that actual software components can be designed. The engineering standards that are going to be adopted and the design trade-offs that are going to be faced within the implementation process of the project will also be defined in this report.

1.1 Object Design Trade-Offs

Functionality vs. Usability: Kebap Tycoon will be a multi-platform game. However, when we consider the size of the screen for Android or iOS platforms, we have to design the user interface in a way that users can see the functions (i.e buttons, menus etc.) easily. Thus, when the number of functions increase, the usability of the game decreases. To resolve this trade-off, we have chosen usability and optimized functionality by using sub-menus under the screen.

Buy vs. Build: In the beginning of the project, we decided to use Parse for the backend services. However, a few weeks ago, we got an e-mail from Parse saying that the service will be fully retired on January 28, 2017. Thus, we had to decide on whether buying the Parse service temporarily until this date or to build our own backend service with a heavy development cost. Considering our long-term goals, we decided to implement our own backend service instead of using Parse in the short term.

Development Cost vs. Functionality: While implementing the login system, we realized that using Google login will bring up a considerable development cost. Thus, we decided to use Facebook login and compromise on functionality.

Understandability vs. Cost: As we decided to use our own backend service, the documentation of the backend is important for the frontend developers. Instead of checking the backend code for a necessary function, we decided to increase understandability using proper backend documentation although it brings time cost. In addition, writing clear and modular code is important for the test phase and maintainability. Thus, with a time cost, all codes will be written in a clear way and understandability will be increased.

Security vs. Cost: Playing Kebap Tycoon requires authorization of the players. Together with this, all information related to game is kept in remote database. Thus, security is an important issue for our application. To increase security, we have to use SSL certificate and constantly check login credentials during the REST communications. This operations however, brings extra cost on development and budget.

Accessibility vs. Maintainability: Our game will be multi-platform to increase accessibility. However, multi-platform applications requires heavy workload because for the updates, development team has to make changes on every platform. Although we use LibGDX to resolve this problem, we need to break our code base for some operations and implement for different platforms one by one. (e.g. login, advertisements etc.)

1.2 Interface Documentation Guidelines

For the class interface documentation, the following style is used:

Class Name	The name of the class
Description	A short description about the class.
Package	The name of the package that includes the class
Attributes	Attributes together with their types that enclosed in the class
Operations	The list of operations together with their return types that can be performed using the class.
Class Interactions	List of classes that interact with this class, if any.

1.3 Engineering Standards

In our project The Unified Modeling Language, **UML** [1], is used for the graphical notations. While describing the designs concerning application UML is a widely accepted engineering standard since with the help of UML, visual models of the object-oriented software system can be created in a more simplified and understandable way. Additionally, **IEEE Citation Style Guide**[2] is used while creating the reports in order to make citations in a standardized way as described in this engineering guide.

1.4 Definitions, Acronyms, and Abbreviations

The following list shows the acronyms and abbreviations used:

•**IEEE:** Institute of Electrical and Electronics Engineering

•**UML:** Unified Modeling Language

•**MVC:** Model -View-Controller

• **SQL:** Structured Query Language

• **UI:** User Interface

• **REST:** Representational State Transfer

2. Packages

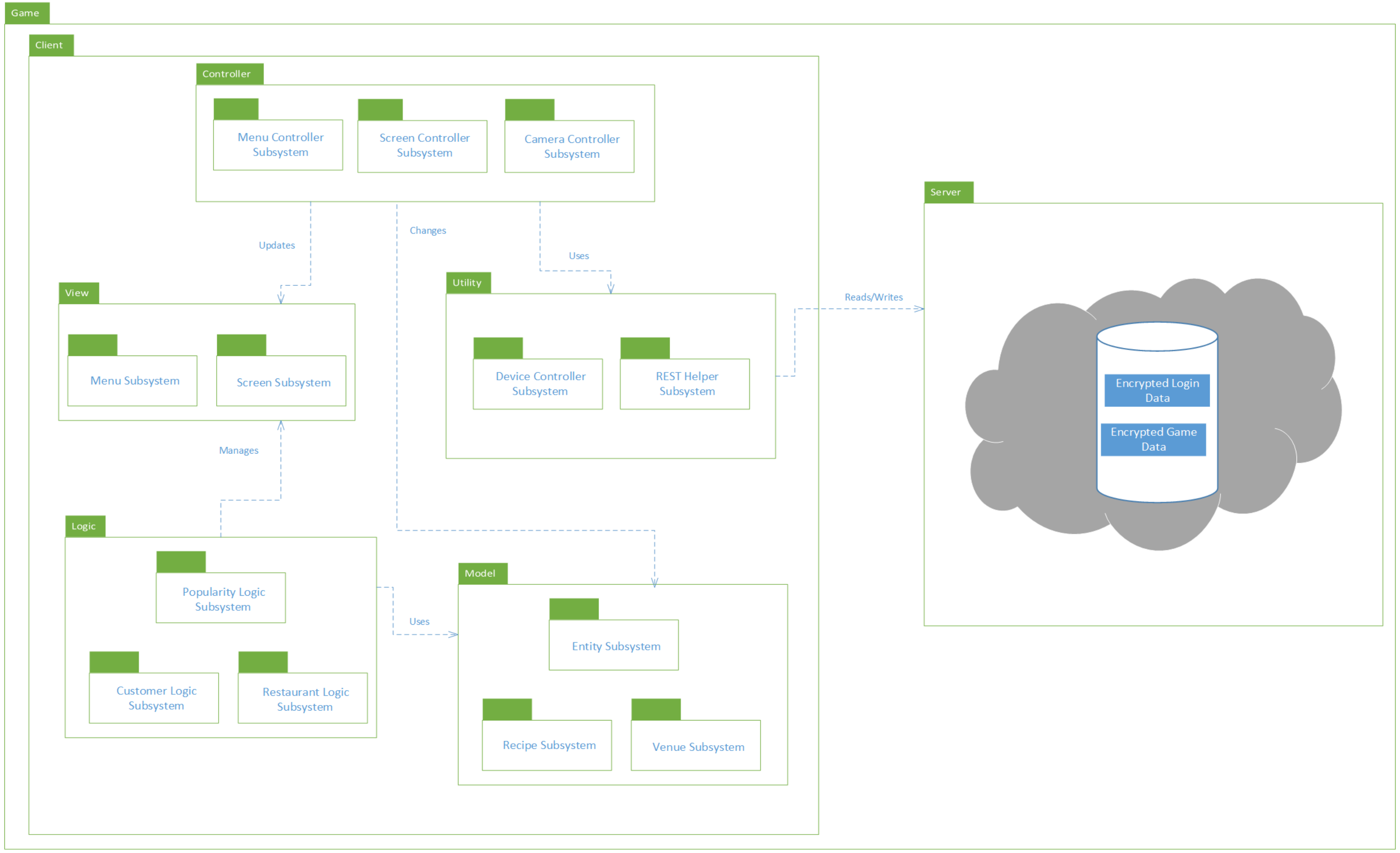

Figure 1 : Subsytem Decomposition Diagram

The subsystem decomposition together with the enclosed packages shown in Figure 1 which is taken from our high-level design report with small changes. As we decided our packages earlier in high-level design phase, with a small change (using our backend service instead of Parse), the implementation and low-level design became easier.

Kebap Tycoon uses MVC approach with some helper packages. Model package includes all entity classes with recipe and venue subsystems which will manage the players' own recipe and venues. View package includes necessary menu and screen classes. Controller package, on the other hand, includes all the classes that controls data flow, user inputs, screens etc. In addition to these packages, we have Utility package that is used as helper for the communication between client and server and Logic package which includes artificial intelligence algorithms to manage popularity, customers and restaurant logic. A detailed information about packages and our decomposition scheme can be found in our high level design report.

3. Class Interfaces

NOTE: All classes have getters and setters, but for the sake of simplicity, they are not included in tables.

3.1 Model Package

Class Name	Entity
Description	This class is an abstract class which represents all entity classes
Package	Model
Attributes	-x: int -y: int -standardAnimation: Animation
Operations	+getCurrentFrame(): GameTexture

Table 1 : Entity class interface

Class Name	Person
Description	Representation of person in the game, extends Entity
Package	Model
Attributes	-state: int -walkAnimation: Animation -speed: int
Operations	+findPath(): Point[]

	+think(): void

Table 2 : Person class interface

Class Name	Furniture
Description	Representation of furnitures in the game, extends Entity
Package	Model
Attributes	-coveredArea: int -price: int -orientation: int
Operations	

Table 3 : Furniture class interface

Class Name	Customer
Description	Representation of customers in the game, extends Person
Package	Model
Attributes	-type: int -age: int -satisfaction: int
Operations	

Table 4 : Customer class interface

Class Name	Employee
Description	Representation of employee in the game, extends Person
Package	Model
Attributes	-experience: int -level: int -MAX_LEVEL: int
Operations	

Table 5 : Employee class interface

Class Name	Recipe
Description	Representation of a specific recipe created by player, extends Entity
Package	Model
Attributes	-MIN_LEVEL: int -ingredients: IngredientTuple[]
Operations	

Table 6 : Recipe class interface

Class Name	Player
Description	Representation of real player that plays the game
Package	Model
Attributes	-level: int -experience: int -userName: String -friends: Player[] -venues: Venue[] -recipes: Recipe[] -money: Double
Operations	+seeFriends(): void +saveScore(): void

Table 7: Player class interface

Class Name	Venue
Description	Representation of venues of players in the game
Package	Model
Attributes	-furnitures: Furniture[] -name: String -popularity: int -rent: int -openingTime: Date -closingTime: Date -stock: IngredientTuple[]
Operations	

Table 8: Venue class interface

Class Name	Report
Description	Representation of reports generated for players to see their revenues, profits, loss etc.
Package	Model
Attributes	-totalDailySales: int -totalCustomers: int -totalConsumption: IngredientTuple[] -rent: int -advertisement: int -profit: int
Operations	+toString(): String

Table 9: Report class interface

Class Name	Restaurant
Description	Representation of restaurant that player owns, extends Venue
Package	Model

Attributes	-kitchenSize: Point
Operations	

Table 10 : Restaurant class interface

3.2 Controller Package

Class Name	Controller
Description	An abstract class for representation of all controller classes
Package	Controller
Attributes	-isActive: boolean
Operations	+ keyDown(int keycode): boolean + keyUp(int keycode): boolean + keyTyped(char character): boolean + touchDown(int screenX, int screenY, int pointer, int button): boolean + touchUp(int screenX, int screenY, int pointer, int button): boolean + touchDragged(int screenX, int screenY, int pointer): boolean + mouseMoved(int screenX, int screenY): boolean + scrolled(int amount): boolean + create(): void + tap(float x, float y, int count, int button): boolean + longPress(float x, float y): boolean + fling(float velocityX, float velocityY, int button): boolean + pan(float x, float y, float deltaX, float deltaY): boolean + zoom(float initialDistance, float distance): boolean

Table 11: Controller class interface

Class Name	DishSelectionController
Description	Responsible for dish selection, extends Controller
Package	Controller
Attributes	
Operations	+selectDish(int dishType): void

Table 12 : DishSelectionController class interface

Class Name	RegisterController
Description	Responsible for player's register operations, extends Controller

Package	Controller
Attributes	
Operations	+registerWithFacebook(): boolean +registerUser(String userName, String password, String email): boolean

Table 13: RegisterController class interface

Class Name	LoginController
Description	Responsible for login operations, extends Controller
Package	Controller
Attributes	
Operations	+login(String username, String password): boolean +loginFacebook(): boolean

Table 14 : LoginController class interface

Class Name	MenuController
Description	Responsible for controlling current menus and flows, extends Controller
Package	Controller
Attributes	
Operations	+convertMenu(int x, int y): Menu

Table 15 : MenuController class interface

Class Name	GameScreenController
Description	Responsible for controlling current screens and flows, extends Controller
Package	Controller
Attributes	
Operations	

Table 16 : GameScreenController class interface

Class Name	MainMenuController
Description	Responsible for controlling main menu and starting game, extends Controller
Package	Controller
Attributes	
Operations	+startGameMenu(): void

Table 17 : MainMenuController class interface

3.3 View Package

Class Name	Menu
Description	An abstract class used for representation of all menus
Package	View
Attributes	
Operations	+displayMenu(): void +createController():MenuController

Table 18 : Menu class interface

Class Name	ShoppingMenu
Description	Representation of shopping menu, extends Menu
Package	View
Attributes	
Operations	

Table 19 : ShoppingMenu class interface

Class Name	EmployeesMenu
Description	Representation of employees menu, extends Menu
Package	View
Attributes	
Operations	

Table 20: EmployeesMenu class interface

Class Name	ScreenFactory
Description	Responsible for creating screens
Package	View
Attributes	
Operations	+createScreen(int screenType):

Table 21: ScreenFactory class interface

Class Name	BaseScreen
Description	An abstract class that represents all screens, implements Screen(LibGDX class)
Package	View
Attributes	-batch: SpriteBatch

	-font: Texture -gd: GestureDetector
Operations	+dispose() : void +hide(): void +pause(): void +render(): void +resize(): void +resume(): void +show(): void

Table 22 : BaseScreen class interface

Class Name	MainMenuScreen
Description	Represents the screen of main menu, extends BaseScreen
Package	View
Attributes	
Operations	

Table 23 : MainMenuScreen class interface

Class Name	SplashScreen
Description	Represents the splash screen, extends BaseScreen
Package	View
Attributes	
Operations	

Table 24: SplashScreen class interface

Class Name	LoginScreen
Description	Represents the login screen, extends BaseScreen
Package	View
Attributes	
Operations	

Table 25 : LoginScreen class interface

Class Name	TutorialScreen
Description	Represents the tutorial screen, extends BaseScreen
Package	View
Attributes	
Operations	

Table 26: TutorialScreen class interface

Class Name	RegisterScreen
Description	Represents the registration screen, extends BaseScreen
Package	View
Attributes	
Operations	+displayMessage(String message): void

Table 27: RegisterScreen class interface

Class Name	GameScreen
Description	Represents the game screen, extends BaseScreen
Package	View
Attributes	-menuStack: Stack<Menu> -controllerStack: Stack <Controller> -currentMenu: Menu
Operations	+createMenu(int menuType): void +closeMenu(): boolean

Table 28: GameScreen class interface

Class Name	Animator
Description	Responsible for the animations of sprites
Package	View
Attributes	-animation: Animation -frames: TextureRegion[] -currentFrame: TextureRegion -stateTime: float -batch: SpriteBatch
Operations	+setupAnimation(SpriteBatch b, int x, int y, TextureRegion[] tr): void +startAnimation(): void +stopAnimation(): void

Table 29: Animator class interface

Class Name	GameTexture
Description	Responsible for initializing and maintaining all sprites
Package	View
Attributes	-textures: Texture[]
Operations	+loadMenu(): void +loadGameScreen(): void +loadSplashScreen(): void +loadLoginScreen(): void +loadRegisterScreen(): void

Table 30: GameTexture class interface

3.4 Utility Package

Class Name	DeviceController
Description	Responsible for input/outputs and Internet connection
Package	Utility
Attributes	
Operations	+isInternetConnected(): boolean +saveLocalData(): boolean +deleteLocalData(): boolean +load(): boolean

Table 31: DeviceController class interface

Class Name	BackendController
Description	Responsible for interactions between client and server
Package	Utility
Attributes	-url : URL -connection: URLConnection -app_id: String -app_key: String
Operations	+storeGameState(): void + storePlayerData (): void +login(String username, String password, String securityOptions): boolean + loginFacebook(): boolean +retrieveGameState() : String[] + retrievePlayerData() : String[] + register(String username, String password, String email): boolean

Table 32: BackendController class interface

Class Name	SocialController
Description	Responsible for Facebook related operations (i.e finding friends)
Package	Utility
Attributes	-url : URL -connection: URLConnection -app_id: String -app_key: String
Operations	+getFriends(String accessToken): String[] +getProfilePicture(String accessToken): String +getUserProfilData(): String[]

Table 33: SocialController class interface

3.5 Logic Package

Class Name	GameEngine
Description	Responsible for all game logic
Package	Logic
Attributes	
Operations	+saveGameData(): void +createEvent(int type): Event +createReport(): Report +getInstance(): GameEngine -GameEngine(): GameEngine +pauseGame(): void

Table 34 : GameEngine class interface

Class Name	CustomerLogic
Description	Responsible for all customers
Package	Logic
Attributes	
Operations	+handleCustomers(): void +findPath(Customer c, target t): int[] +evaluateFood(Customer c, Dish d): int

Table 35 : CustomerLogic class interface

Class Name	PopularityLogic
Description	Responsible for all popularity of the restaurants
Package	Logic
Attributes	
Operations	+handlePopularity(Venue v): void +getTeenagerScore(Venue v): int +getAdultScore(Venue v): int +getSeniorScore(Venue v): int

Table 36 : PopularityLogic class interface

4. Glossary

LibGDX: Game-development application framework written in the Java programming language. It allows for the development of desktop and mobile games by using the same code base [3].

REST: Representational State Transfer

5. References

[1] Uml.org, "Unified Modeling Language (UML)", 2016. [Online]. Available: http://www.uml.org/. [Accessed: 13- Feb- 2016].

[2] Ieee.org, "IEEE Publications & Standards", 2016. [Online]. Available: http://www.ieee.org/publications_standards/index.html. [Accessed: 13- Feb- 2016].

[3] Libgdx.badlogicgames.com, "libgdx", 2015. [Online]. Available: http://libgdx.badlogicgames.com. [Accessed: 13-Feb-2015].