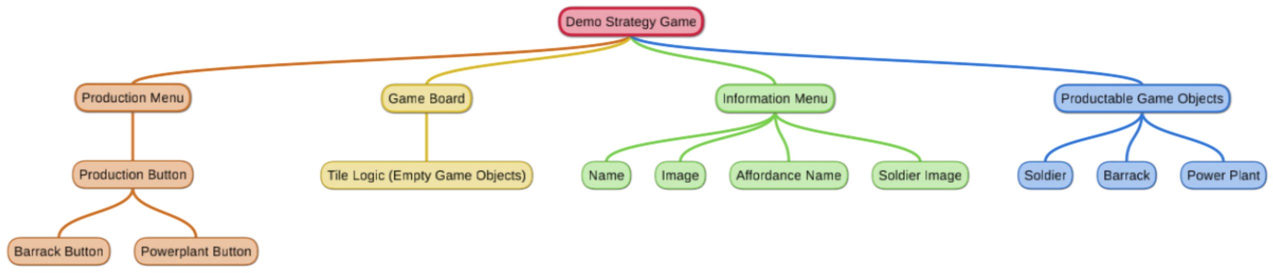
Demo Strategy Game Design Document



SelectingObject: This selection logic is created with raycast from mouse point. There is a ray from camera through the mouse point. If it is collided with any selected game object, then it sets the selecting logic true. Therefore, if player clicks anywhere else rather than on the selectable object; then s/he automatically deselect selected object. This script is on the **"GameManager"** object.

Resolutions: 4:3 and 16:9

1. **"Production Menu"** is empty ~~sprite~~. It consists two different sprites called "~~**Barrack Button**~~" and "~~**Power Plant Button**~~". There are more buttons than rendered buttons on the screen. This empty ~~sprite~~ has **"ProductionMenu"** script. In this script, here are **"Scroll"**, **"CreateButtonTop"**, and **"CreateButtonBottom"** functions; with **"isCrossedBottom"** and **"isCrossedTop"** variables. Production menu is scrollable. This scrolling action is detected ~~if player clicks and holds empty space in Production Menu~~. Only Y position of "buttons" are affected.

bool isCrossedBottom: True if bottommost buttons crossed the bottommost threshold line. Indicates that topmost buttons' positions should be moved to the desired bottommost position below the threshold line. Threshold lines are vertical borders of main camera.
bool isCrossedTop: The exact opposite of **"isCrossedBottom"** variable, with same threshold lines.

~~**Scroll:** As I indicated, if player clicks and holds left mouse button, all buttons' (barrack and powerplant buttons) Y positions are changed with mouse's Y position with offset value. So that, player smoothly scrolls these buttons.~~ This scrolling action is achieved by object pooling.
CreateButtonTop: This function is triggered with **"isCrossedTop"** variable. It gets the bottommost buttons and moves them to the topmost desired location. As touched, there are more buttons that player can see on the screen, so changing the topmost or bottommost buttons will not be visually noticed by players.
CreateButtonBottom: This is the exact opposite of **"CreateButtonTop"** function.

1.1. **"Barrack Button"** and **"Power Plant Button"** are created with inheritance from base abstract class called **"Production Button"**. This abstract class consists functions called **"IsSelected"**, **"OnDragged"**, **"OnBuilt"**, **"IsBuildable"**, **"ShowBuildArea"**, and **"MarkCells"**. This abstract consists variables of **"canBeDragged"**, **"oldPosition"**.

bool canBeDragged: True if button is clicked. If somewhere else is clicked, then it is false.
~~**bool**~~ **oldPosition:** vector2 variable. Holds position of button before dragging action.

IsSelected: This function continuously checks **"Selecting Object"** class to set **"canBeDragged"** true or false. ~~This button also changes Information Menu's **"string**~~

~~**selectedItem"** component with **"Soldier"**, **"Barrack"**, or **"Power Plant"** according to button.~~

OnDragged: This function is called when button is dragged. Function both detects dragging action and mouse release action to call **"IsBuildable"** function. Dragging action first checks if **"canBeDragged"** variable is true. Then function sets button's position to mouse's position with offsets for smooth sliding. **"ShowBuildArea"** function is also called in this function for affordance concern.

OnBuilt: This function is called when mouse is released (with barrack or powerplant button) on the game board. **"IsBuildable**" function is in this function. If area is buildable, then product is built with instantiate function calling prefab; and used cells are marked. This mark action is achieved by sending all subscribers (cells) an event and telling them they are now busy. At the every end of **"OnBuilt"** function, button is returned to the **"oldPosition"**.

MarkCells: This function is event, and listened by all subscribers (cells).

IsBuildable: This function checks if dragged object is fit with game tiles on the board or not. This function looks all cells in the built area and checks if all areas are empty; returns false if even one of cells is busy.

ShowBuildArea: This function is created due to affordance concern. Built area (4x4 cell for barrack, 2x3 cell for powerplant) is shown as rectangle around the dragged button on the fly.

1.1.1. **"Barrack Button"** is mainly used to create barracks.
1.1.2. **"Power Plant Button"** is mainly used to create power plants.

2. **"Game Board"** is 15x10 cell-sized empty sprite. This sprite serves as a container for game tiles (cells). This sprite is only used for its transform component.

2.1. **"Tile Logic"** is implemented using empty square (32x32 px) sprites, which are also called as **"cells"**. Cells have **"cell"** script. In this script; there is **"isBusy"** variable.

bool isBusy: This is private variable encapsulated with public function. This function subscribes deployment method. tells us if the tile is used with something like soldier or some part of building.

3. **"Information Menu"** is rightmost UI element. It has **"~~ChangeInformation~~"** script to change its components. This menu has components of **"Name"**, **"Image"**, **"Affordance Name"**, and **"Soldier Image"(Button)**. Inside ChangeInformation script, there is **"selectedItem"** variable; and **"ChangeComponents"**, and **"DeploySoldier"** functions.

string selectedItem: This string can take three different values: **"Soldier"**, **"Barrack"**, and **"Power Plant"**. Its property is checked and changed inside the Update method by looking at **"SelectingObject"** class.

ChangeComponents: This function is inside Unity's Update function, and has if-elseif-elseif blocks:

If **"Barrack"**; then Name: BARRACKS, Image: BARRACKS SPRITE, Affordance Name: PRODUCTION, Soldier Image(Button): SOLDIER SPRITE.

Else if **"Soldier"**; then Name: SOLDIER, Image: SOLDIER SPRITE, Affordance Name: NONE, Soldier Image(Button): NONE.

Else if **"Power Plant"**; then Name: POWER PLANT, Image: POWER PLANT SPRITE, Affordance Name: NONE, Soldier Image(Button): NONE.

DeploySoldier: This function is called when **Soldier Image (Button)** is clicked. It firsts randomly chooses one of built barrack(s) (if any). Then if any barrack is chosen, it randomly chooses one of **not busy** tiles around the barrack that is available for deploying soldier. Then instantiates soldier prefab in this cell. Lastly marks this cell as busy.

4. There are three **"Productable Game Objects"** in the game; which are **"Soldier"**,

“Barrack”, and **“Power Plant”**. They are prefabs which are instantiated on the Game Board.

4.1. **“Barrack”** is used to deploy soldiers.

4.2. **“Power Plant”** is only cosmetical object.

4.3. **“Soldier”** can be moved (if s/he has a path to the destination) to the destination with left mouse click. A* is used as Pathfinding algorithm. Therefore, soldier has a script named **“AStarPathFinding”**.

4.3.1. Manhattan Distance will be used for **“heuristic” (h)**. This is a guess.

4.3.2. Distance between neighbors are used to calculate **“g”**. This can be thought as exact cost for the next step. https://www.geeksforgeeks.org/a-search-algorithm/

Umut EFİLOĞLU