UI Interface:

Buidling From User perspective

START

1. Clicking “Build” UI Button : (Named ‘Building Button’ )

* trigger ‘BuildingUI’ GO .
* trigger ‘BuildingUI’ GO’s phase 1 GO and disable phase 2 GO(for refreshing).

2. Choosing From Phase 1 options (options :Farms,Barracks,other)

Each option does:

* Trigger Phase 2 GO with chosen Phase 2 options GO (for Barracks: Archers ,infantry etc.., option enable).
* Disable other phase 2 options which are not chosen. (like farm sub option: stone,grain.).

3. Choosing From Phase 2 options (Mage ,Infantry,etc,):

* Calls BuildingUIManager GO’s BuildingOptionFunctions Cs’s \_\_\_IsChosen() F. (\_\_\_ is filled with chosen option for Mage : MageIsChosen() F).
* BuildingUI also disables by Cs.

4. A ‘ConstructionInterface’ GO’s CheckingUp GO pops up with the costs and ‘Build’ button option for confirmation.

5. Clicking ‘Build’ button ‘CheckingUp’ GO disables and goes in two case :

a. If we have enough credits (we proceeds )

b. If we don’t have enough credits ,we get message of not enough credit(3 sec) and we Cancels whole process ,Flow tree finishes.

6. If we proceeds, a Blueprint of Building is created with Construction interface . with two option .

a. Yes: Check if blueprint have enough space .Cases:

* Yes: Blueprint is destroy and replaced with building .
* No: a message pops up of not enough space.

b. No: Cancel whole process and destroy blueprint.

Independent Mech:

A. Blueprint turns red when it doesn’t have enough space.and Move with screen.

END

Process in little detail:

The complex Process starts with step 3 of ui with calling of a Cs function .

BuildingUIManager GO’s BuildingOptionFunctions Cs’s StoneFarmIsChosen() F //this Cs includes hand written options link //manually with ui buttons.

*>> BuildingOptionFunctions : MonoBehaviour*

public void StoneFarmIsChosen{

buildingUIManager.BuildingCostInit(buildingManager.BuildingChosen("StoneFarm"));

}

*>>BuildingManager : MonoBehaviour*

public BuildingCost BuildingChosen(string buildingName, int level=1){

//this will be called by uimanager

buildingCost=statsManager.GetBuildingStats(buildingName,level);

AssigningCostBM();

return buildingCost;

}

*>>BuildingStatsManager : MonoBehaviour*

[SerializeField] private BuildingData woodFarmData,…,, all building; // Drag the BuildingData //ScriptableObject here

public BuildingCost GetBuildingStats(string buildingName, int levelNumber){

// Ensure the levelNumber is valid (between 1 and 30)

if (levelNumber < 1 || levelNumber > 30)

{

Debug.LogError("Level must be between 1 and 30");

return null;

}

BuildingData buildingData = null;

// Find the correct building based on its name

if (buildingName == "WoodFarm")

{

buildingData = woodFarmData; // Reference to the ScriptableObject containing Wood Farm data

}

else if.

….

else{Debug.LogError("Building not found: " + buildingName);

return null;

}

// Adjust level number to index (array starts at 0, levels start at 1)

int levelIndex = levelNumber - 1;

// Return the resource costs for the specified level

return new BuildingCost( //this one is in resource Spawner

buildingData.woodUpgradeCost[levelIndex],

buildingData.grainUpgradeCost[levelIndex],

buildingData.stoneUpgradeCost[levelIndex],

buildingData.timeCost[levelIndex],

buildingData.BuildingBlueprint,

buildingData.buildingPrefab

);}

*>>BuildingCost*

//this one is kind a payload structure to pass value around .

using UnityEngine;

[System.Serializable]

public class BuildingCost

{

public int woodCost,grainCost, stoneCost, timeCost;

public GameObject TheBlueprint,TheOriginal;

public BuildingCost(int wood, int grain, int stone,int time, GameObject Blueprint,GameObject Original){

woodCost = wood;

grainCost = grain;

stoneCost = stone;

timeCost= time;

TheBlueprint=Blueprint;

TheOriginal=Original;

}}

>>BuildingManager

private void AssigningCostBM(){ \*\*\*\*\*

if(buildingCost==null){

Debug.Log("null buildingcost");

return;

}

woodCost=buildingCost.woodCost;

grainCost=buildingCost.grainCost;

stoneCost=buildingCost.stoneCost;

buildingPrefab=buildingCost.TheOriginal;

buildingBlueprint=buildingCost.TheBlueprint;

}

*>>BuildingUIManager : MonoBehaviour*

public void BuildingCostInit(BuildingCost BuildingCost){

buildingCost=BuildingCost; //Storing buildingCost in Cs

AssigningCost();

CheckingUpUIGameobject.SetActive(true); //Costs UI GO

BuildingUIPanel.SetActive(false); //Building Panel ‘BuildingUI’ GO

DisplayingDataUI();

}

void AssigningCost(){ //assigning stored Cost to separately.

woodCost=buildingCost.woodCost;

grainCost=buildingCost.grainCost;

stoneCost=buildingCost.stoneCost;

}

void DisplayingDataUI(){ //this display cost in ‘CheckingUpUIGameobject’ GO UI

//this will display costdata of that building.

woodsCostUI.text = "W:" + woodCost.ToString() ;

stoneCostUI.text = "G: " + stoneCost.ToString();

grainCostUI.text = "S: " + grainCost.ToString() ;

}

Step 4 done too

Summary:

Each option of building button is linked to a function 1 .which calls a function to get stats and store of that building. Function 2 also stores data and display cost to user with confirmation to build.

Step 5 clicking Building Button ‘Build’ GO’s button in CheckingUp GO’s family.

That disable ‘CheckingUp’ GO and trigger a F

*>>BuildingUIManager : MonoBehaviour*

public void BuildOptionClicked(){

//triggered by ui build button

//check isenough

if(!buildingManager.IsEnoughCredit()){ //Step 5b.

MessageForNotEnoughCredit();

status=3;

RevertingUI(); }

else{ //step 5a

//this for creating blueprint

buildingManager.BuildStage2();

//trigger confirmationui ,the one with tick and cross (last ui panel)

ConfirmationUI.SetActive(true); //step 6 starts here

}

*>>BuildingManager : MonoBehaviour*

public bool IsEnoughCredit(){

return conditionManager.CheckIsEnough(woodCost,grainCost,stoneCost); }

>> ConditionalManager : MonoBehaviour

public bool CheckIsEnough(int woodCost,int grainCost,int stoneCost){

return tradingManager.IsEnoughResource( woodCost, grainCost, stoneCost); }

>> TradingManager : MonoBehaviour

public bool IsEnoughResource(int woodCost,int grainCost,int stoneCost){

//getting all the resources

allResources = currencyManager.ReturnAllResources();

//checking all the resources ++++++++++

//might return some numbers for ui missing resource counter

return allResources[ResourceType.Wood] >= woodCost &&

allResources[ResourceType.Grain] >= grainCost &&

allResources[ResourceType.Stone] >= stoneCost; }

>> CurrencyManager : MonoBehaviour

private Dictionary<ResourceType, int> resourceCurrencies = new Dictionary<ResourceType, int>()

{ { ResourceType.Wood, 0 },

{ ResourceType.Grain, 0 },

{ ResourceType.Stone, 0 } };

public Dictionary<ResourceType, int> ReturnAllResources() {

return new Dictionary<ResourceType, int>(resourceCurrencies); // Return a copy of the dictionary }

public enum ResourceType

{ Wood,

Grain,

Stone}

//step 5b

>> BuildingUIManager : MonoBehaviour

void MessageForNotEnoughCredit(){

NotEnoughCreditsGameobject.SetActive(true);

//create a function to turn it false after 1 sec.

Invoke("HideNotEnoughCreditsMessage", messageDisappearingTime); }

// This method will deactivate the GameObject

void HideNotEnoughCreditsMessage()

{//this is getting invoked by messageFornotenoughcredit.

NotEnoughCreditsGameobject.SetActive(false); }

…..

public void RevertingUI(){

ConfirmationUI.SetActive(false); //This one is yes or no UI GO ’Confirmation’ of ‘ConstructionInterface’ ,the one used to confirm placing

CheckingUpUIGameobject.SetActive(false); //This one displays cost GO ‘CheckingUp’

}

This case ends here and Tree stops.

Summary:

Clicking build option checks the enough credit .In This not credit is executed. UI is reverted with a message. Tree stops.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//step 5a

>>BuildingManager

public void BuildStage2(){ //this will be triggered by build option ,when enough credit available

conditionManager.SpawningBluePrint(buildingBlueprint); }//buildingBlueprint is already stored in buildingManager in step 3 [\*\*\*\*\*](#StoringBuildingDatainBM)

>>ConditionalManager

public void SpawningBluePrint(GameObject chosenBlueprint){

//instiate blueprint and assign them

TheChosenBlueprint=Instantiate(chosenBlueprint);

bluePrint=TheChosenBlueprint.GetComponent<BluePrint>();

}

Summary:

Enough credit case instantiate the blueprint .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//Transitioning to Step 6

//6a. Clicking ‘Yes’ Button:

Triggers a function in BuildingUIManager GO ‘s BuildingUIManager Cs’ ConfirmationClicked F

>>BuildingUIManager

public void ConfirmationClicked(){ //this will be triggered by tick

status=buildingManager.ConfirmingBuilding();

if(status==0){

Debug.Log("success");

RevertingUI();

nullingCost();

globalUIManager.RefreshPermission(); } ///This will not be explained here

else {

if(status==1){

Debug.Log("Not inside kingdom");

RevertingUI();

nullingCost();

//not inside kingdom }

else if(status==2){ // no refreshing ui here.

Debug.Log("Not Enough space");

MessageForNotEnoughSpace(); } }

//check the status and display a message depends on status }

>> BuildingManager : MonoBehaviour

public int ConfirmingBuilding(){

//triggered by tick ui indirectly

status=conditionManager.CheckAllTheCondition();

if(status==0){

CutTheBuildingCost();

SpawnBuilding();

NullingData(); }

else if(status==1){

conditionManager.DestroyTheBlueprint();

NullingData(); }

return status; }

>>ConditionalManager

public int CheckAllTheCondition(){ //inside innerkingdom

if(!bluePrint.ReturnIsInsideKingdom()){ // inside the kingdom ,This will be added in future.

Debug.Log(bluePrint.ReturnIsInsideKingdom());

return 1; }

else if(bluePrint.ReturnIsColliding()){

return 2; }

return 0; //if no problem }

>>Buidling Manager

private void CutTheBuildingCost(){

tradingManager.SpendingResources(woodCost, grainCost, stoneCost);

}

private void SpawnBuilding(){

SpawnedBuilding=Instantiate(buildingPrefab,conditionManager.GetTheBlueprintPosition(),

Quaternion.identity);

ProvidingManager();

conditionManager.DestroyTheBlueprint();

}

private void NullingData(){

buildingCost=null;

woodCost=0;

grainCost=0;

stoneCost=0;

buildingPrefab=null;

buildingBlueprint=null;

}

>>ConditionalManager

public void DestroyTheBlueprint(){

Destroy(TheChosenBlueprint);

bluePrint=null; }

>>BuildingUIManager

public void RevertingUI(){

ConfirmationUI.SetActive(false);

CheckingUpUIGameobject.SetActive(false); }

private void nullingCost(){

//this will be triggered automatically when the building is done or canceled

buildingCost=null;

woodCost=0;

grainCost=0;

stoneCost=0;

}

public void MessageForNotEnoughSpace(){ //this will be called by RSM

NoSpaceGameobject.SetActive(true);

Invoke("HideNotEnoughSpaceMessage", messageDisappearingTime); }

void HideNotEnoughSpaceMessage()

{ NoSpaceGameobject.SetActive(false); }

Summary:

Confirming checks Status which are based on building blueprint like insidekingdom, colliding. If all condition meets blueprint is destroyed and building is instantiated on it’s place. UI is reverted and data is nulled in the script.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

All Phases Ends

Independent Mech:

Blueprint conditions mech