YPC2102
Group Members:
Lam Puy Yin (1155126240)
Cheng Ka Pui (1155125534)
Title: Serious Game Development I

Project Name: SDG World City

SDG World City: Final Report

Introductions

(Figure 1. SDG goals)

Project Goal:
Inspired by the official SDG 2030 game[1] and other similar card games[2]. We hope that through a city planning game experience, we can let people understand more about the 17 Sustainable Development Goals (SDGs) and reflect on their actions in daily life, becoming aware of the World and themselves in promoting for sustainable development.

Project Background:

- A city planning and environmental skybox game
- Focus on urban development while also have heavy emphasis on international impacts, by adding global crisis in the game, we let player learn about the effect of urban planning, how it can shape the world, and the importance of the 17 Sustainable Development Goals
- Poor planning in city development would affect the livelihood of your city, which will be shown in the end, and let you rethink your city planning approaches as you reaches it…
- Also implemented a global relations system, which can allow players to help contribute other cities in their urban projects, trade deals, and cooperating in improving global cities development, emphasising the importance of global cooperation with careful considerations of cities' needs in order to achieve the 17 Sustainable Development Goals.

Game Goal:
After all turns the world will reach the year 2030. The player shall get a score as high as possible after all turns. Or the player can build their dream world as they wish.

Total Score:
Global index + local index (Both indices have social, environmental, and economic aspects)
Global indices are calculated by the average of all cities' index level (except player's city level)

How to play:

- Use action cards to initiate city projects
- Gain or lose city index from the projects
- Solve global crisis events to prevent global index loss
    - Global events can be solved by 2 ways:
    - If player have a building which its SDG is related to the current issue, they can choose to initiate project research base on that building
    - If player do not own any SDG related buildings, then they can also choose a tile for constructing a temporary building with the related SDG

(Action Cards are based on the 17 Sustainable Development Goals. Link: https://sdgs.un.org/goals)

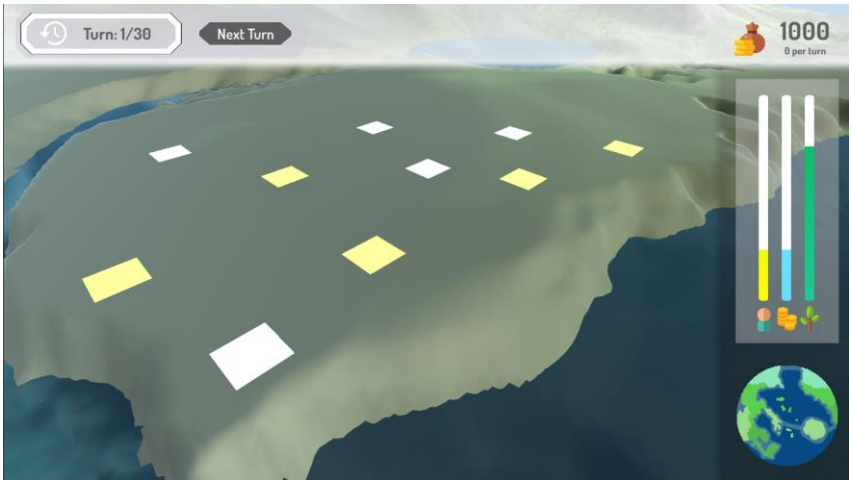
Game Rules

- **Starting rules**
    - Player should choose their own city's location at the first turn (each would have a different index levels in all 3 aspects)

(Figure 2. First turn city selection panel)

    - Players have certain amount of money on their first turn
    - Game will end after ~50 turns, and show player's performance result based on their index score (local and global)

- **In-game rules**



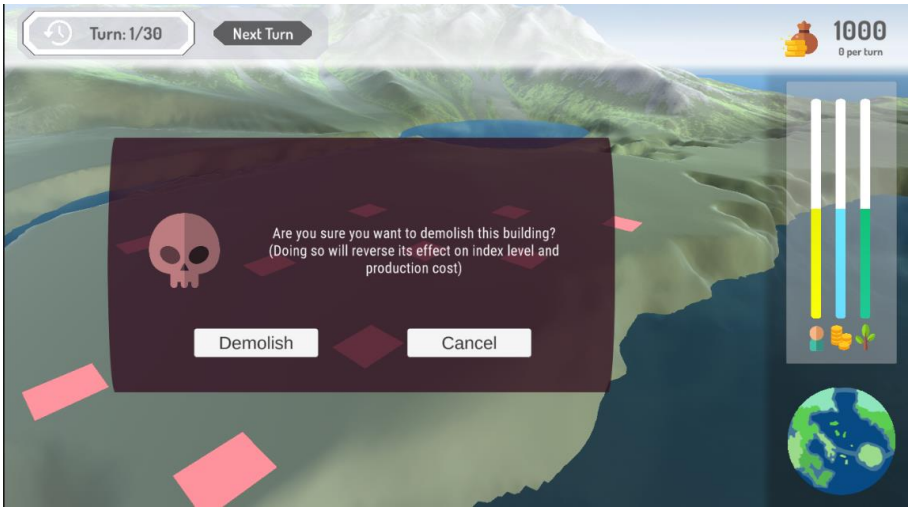
(Figure 3.1. Local city panel)

- **City panel**:

- Each turn player can place a certain amount of action cards
- All action cards are required to take a certain amount of time before completion, which is expressed in the form of in-game turns
- All action cards also require funds to complete the project, which is expressed in the form of in-game money
- Thus emphasising the importance of resources management when developing a city, let player to learn how to maintain a sustainable city with limited spaces, time, and resources

(Figure 3.2. tile purchase)

- Player can buy new tiles for city planning given that they have enough money to do so, tile cost will also increase after every purchase.



(Figure 3.3. building demolish)

- Player can demolish buildings to build other buildings, doing so will reverse its effect on index level and production cost

(Figure 3.4. Global panel)

- **Global panel**:

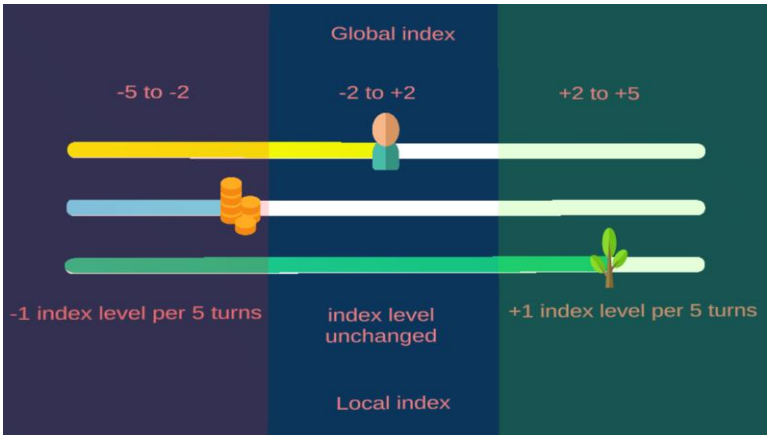
- Cities would be represented as 6 continents, each having different index levels in all aspects
- Players can choose to engage in project exchange with other cities around the world, trade operation can only be performed once in each turn
- Whether the trade is successful is based on the cities' requirements, which is composed of their trading preference and city's index level, and therefore require player's careful planning in trading specific cards that would satisfy the needs of the city in order to perform a successful trading, thus highlighting the importance of global cooperation in pursue for global sustainable development.

(Figure 3.5 Global Crisis)

- Emergencies will occur randomly, players can choose address in the global crisis and help complete goals to advert the crisis before certain turns was reached

(Figure 3.6 Global Crisis occurred)

- Failure in crisis prevention will affect global index negatively
- Global index level will have additional effect on local index level (e.g. if global index is below -2 or above 2, then local index will decrease or increase by 1 in every 5 turns respectively), thus encouraging players to participate more in global events



(Figure 3.7 Global index affecting local index)

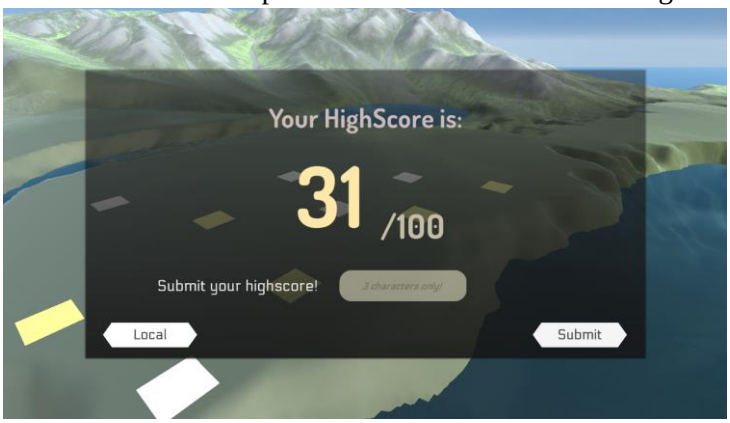
- **End Game**

  - Game will end after 30 turns

(Figure 4.1 Local Ending Page)

(Figure 4.2 Global Ending Page)

  - Player's performance will be represented as local and global scores
  - The final score result is represented as the sum of local and global score



(Figure 4.3 Score Ending Page)

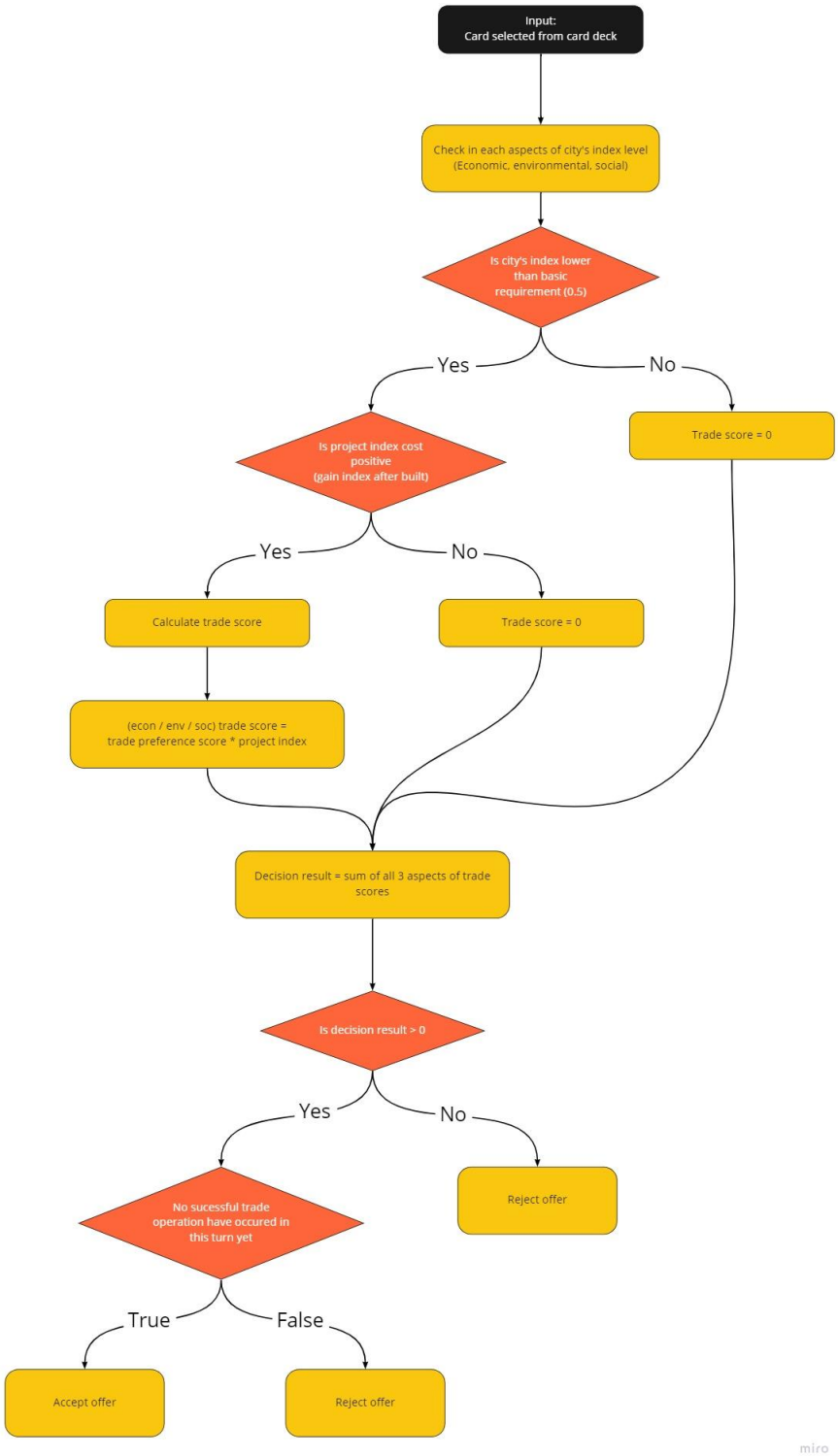
  - Player should input their name (no more than 3 characters) to save their personal record onto the game scoreboard before quitting the game

Game Flow

- A basic Game Flow of the game

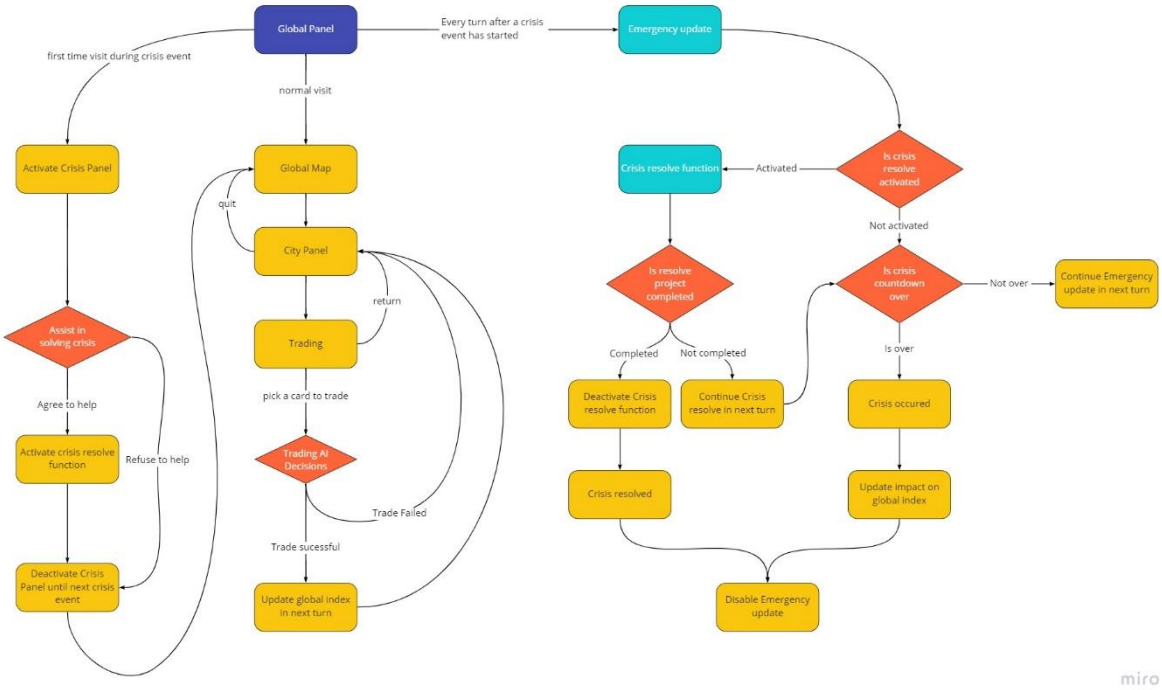
(Figure 5. Game Flow)

- Game AI (AI Card Trading)
    - AI decisions during a trade operation is initiated from the player, it takes the input of card preset which the player is going to trade with, and decides whether the AI should accept this trade offer or not base on multiple conditions as the following (details of the AI is inside Continent_AI from Game Scripts):



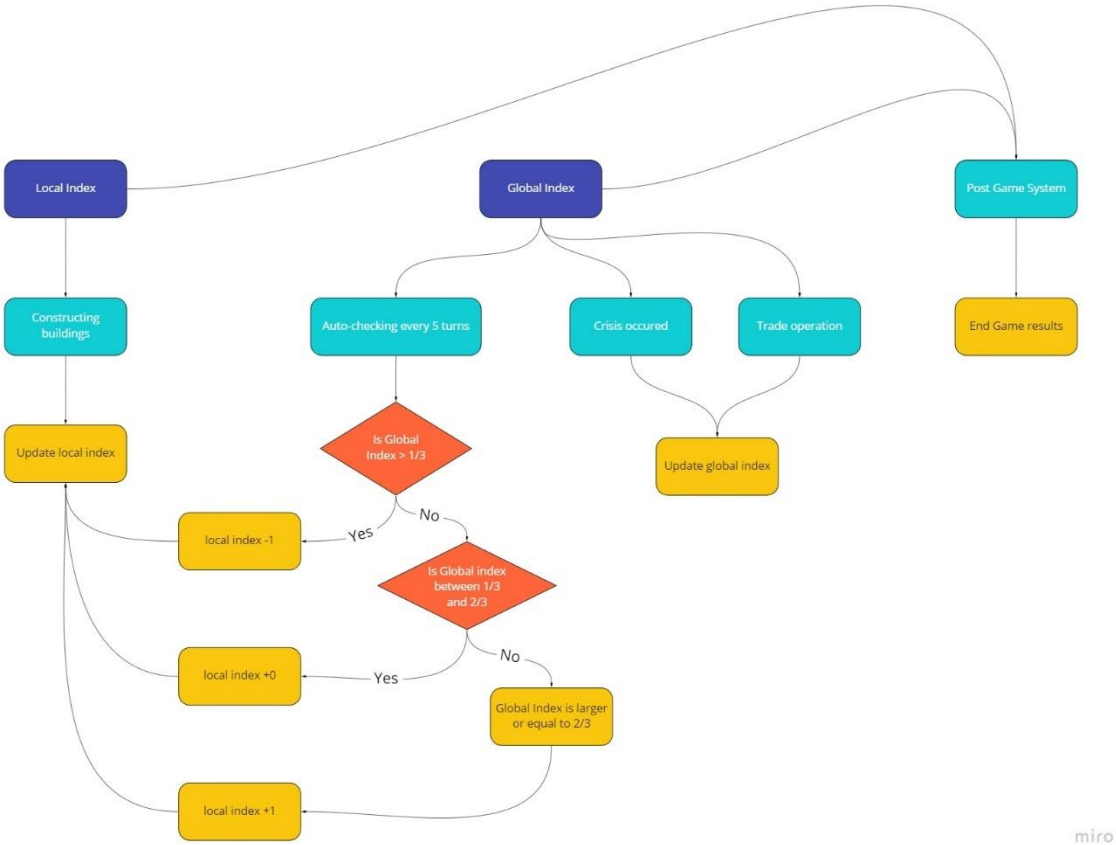
(Figure 6.1. Game AI)

- Global Panel (Global Event System and Crisis Event)
    - Responsible for controlling global panel such as trading and global crisis events in-game (details of the Global Event System and Crisis Event is inside GlobalManager, Global_Crisis respectively from Game Scripts):



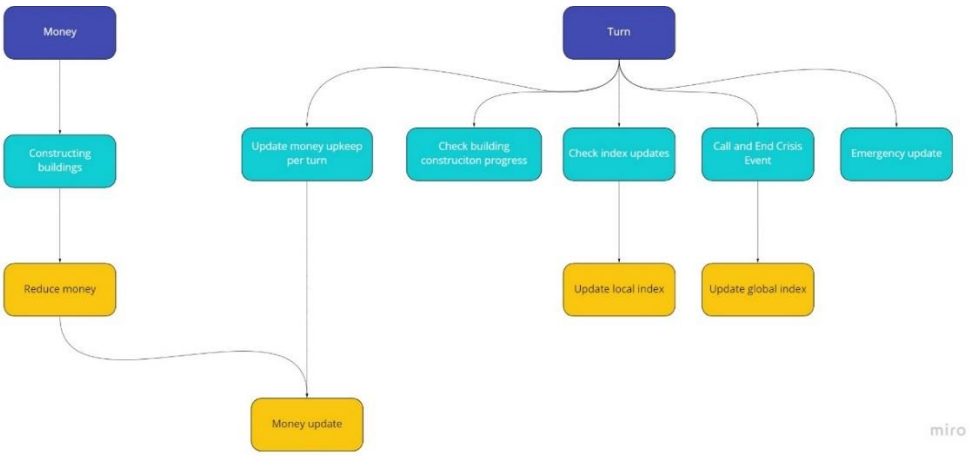
(Figure 6.2 Global Event System and Crisis Event)

- Local and Global Index (Index System)
    - Responsible for controlling the indices (economic, environmental, social index) of global and local through function updates (details of the Index System is inside GlobalManager and game_manager from Game Scripts):



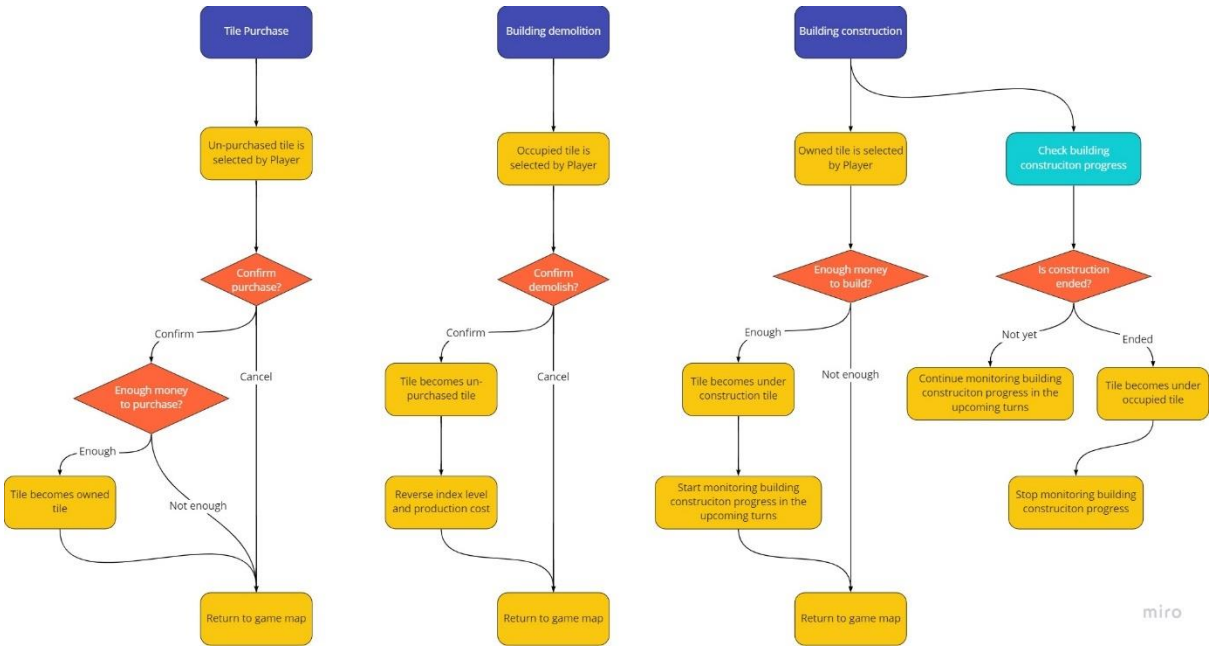
(Figure 6.3 Index System)

- Money and Turn (Resource Management)
    - Responsible for controlling in-game currency and time-duration of the game (details of the Resource Management is inside game_manager from Game Scripts):



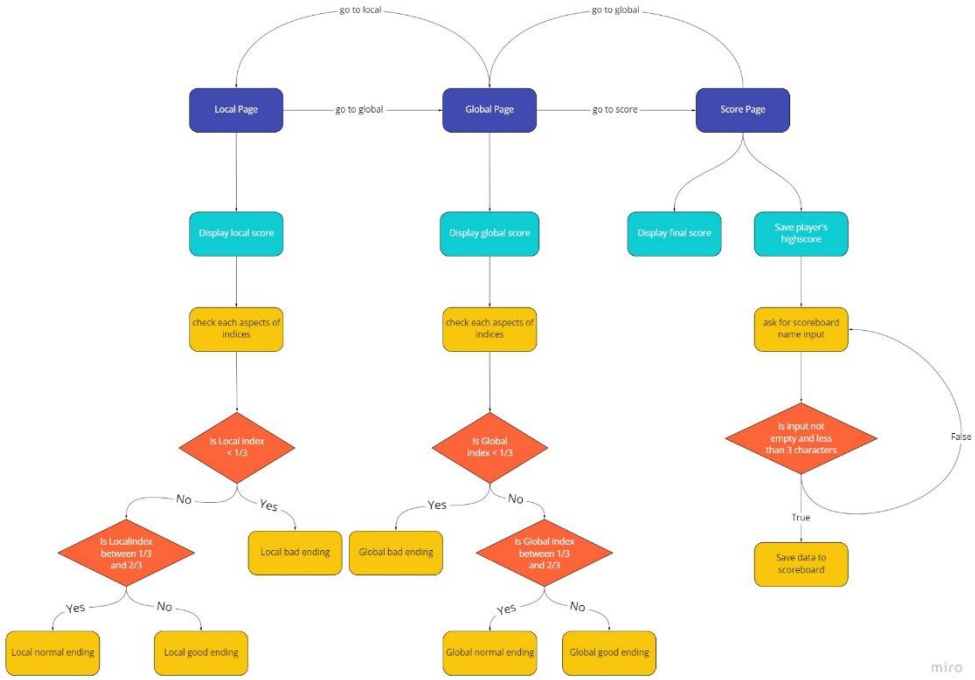
(Figure 6.4 Resource Management)

- Tiles and buildings (Tile Purchase / Demolition and Construction)
    - Responsible for controlling tile purchase, building demolition and construction actions (details of the Tile Purchase / Demolition and Construction is inside game_manager, Tile_placer, and Tile_selector from Game Scripts):



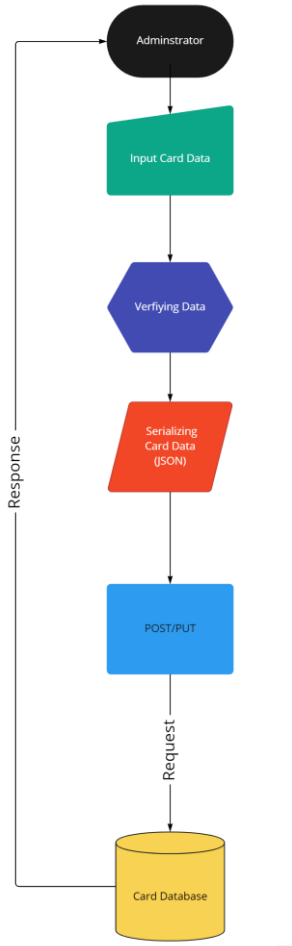
(Figure 6.5 Tile Purchase / Demolition and Construction)

- Ending Scene (Post Game System)
    - Responsible for displaying the all game scores (local, global, and final) according to Player's performance, as well as recording Player's highscore to the scoreboard (details of the Post Game System is inside EndingScene from Game Scripts):



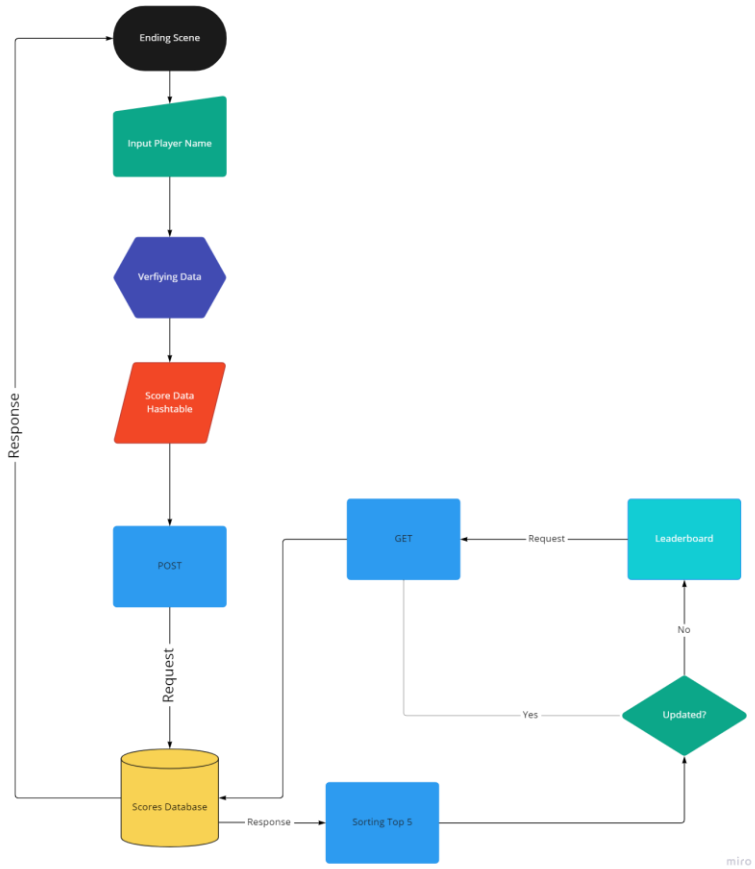
(Figure 6.6 Post Game System)

- Cards (Card System)
    - Responsible for generating cards for project and emergency cards during game (details of the Card System and its API is explained in the report of my other group member):



(Figure 6.7 Card System)

- Scoreboard (Scoreboard System)
    - Responsible for generating a scoreboard after a game has finished (details of the Scoreboard System and its API is explained in the report of my other group member):



(Figure 6.8 Scoreboard System)

Division of Labour

The following parts are what I had worked on and contributed to the project:

1. Game Scripts

Game_manager

Main script for the game, perform tasks from both interface and in-game objects

- Tile construction (and visualisation)
    - Responsible for tile and building related functions, the script will check if the player money is enough for the building cost, if so, it will identify the building position and check if the tile has already been occupied with another building. If all requirements are satisfied, the function will perform updates for the tile, player's city index, and player's treasury

```
public void PlannedBuildings(TileScript buildingPreset)
{
    waitingBuildingPreset = buildingPreset;

    if (playerMoney < waitingBuildingPreset.moneyCost)
    {
        Debug.Log("not enough money!");
    }
    else
    {
        bool isDuplicated = false;
        int currentTileNum = tilePlacer.getSelectedTileNum();
        if (avaliableWaitingSlot[currentTileNum] == false)
        {
            if (plannedPosition[currentTileNum] == tile_placer.inst.placementIndicator.transform.localPosition)
            {
                Debug.Log("tile is already occupied!");
                isDuplicated = true;
            }
        }
        if (isDuplicated == false)
        {
            if (avaliableWaitingSlot[currentTileNum] == true)
            {

                Debug.Log("buildingprefab is:" + waitingBuildingPreset.prefab + " in waitingPrefab: " + waitingPrefab[currentTileNum]);

                waitingPrefab[currentTileNum] = waitingBuildingPreset.prefab;
                waitingTurnCost[currentTileNum] = waitingBuildingPreset.turnCost;
                waitingMoneyCost[currentTileNum] = waitingBuildingPreset.moneyCost;
                UpdatePlayerMoney(-(waitingMoneyCost[currentTileNum]));
                buildingMoneyReturn[currentTileNum] = waitingBuildingPreset.moneyReturn;
                waitingLV[0, currentTileNum] = waitingBuildingPreset.econLV;
                waitingLV[1, currentTileNum] = waitingBuildingPreset.envLV;
                waitingLV[2, currentTileNum] = waitingBuildingPreset.socLV;
                avaliableWaitingSlot[currentTileNum] = false;

                occupiedBuildingPreset[currentTileNum] = waitingBuildingPreset;

                plannedPosition[currentTileNum] = tile_placer.inst.placementIndicator.transform.localPosition;

                tilePlaceholder[currentTileNum] = Instantiate(buildingPlaceholder, plannedPosition[currentTileNum], Quaternion.identity);

            }
        }
    }


    tile_placer.inst.CancelTileSelection();

}

public void CheckPlanning()
{
    for (int currentTileNum = 0; currentTileNum < 10; currentTileNum++)
    {
        if (avaliableWaitingSlot[currentTileNum] == false && occupiedTile[currentTileNum] == false)
        {
            waitingTurnCost[currentTileNum] = waitingTurnCost[currentTileNum] - turnProduction;
            if (waitingTurnCost[currentTileNum] <= 0)
            {
                Destroy(tilePlaceholder[currentTileNum]);
                tile_placer.inst.PlaceBuidling(waitingPrefab[currentTileNum], plannedPosition[currentTileNum], currentTileNum);
                occupiedTile[currentTileNum] = true;

                turnMoney += buildingMoneyReturn[currentTileNum];

                updateIndex(waitingLV[0, currentTileNum], waitingLV[1, currentTileNum],waitingLV[2, currentTileNum]);
            }
        }

    }


}
```

(Figure 7.1 Game_manager script)

- End turn updates
    - For checking and updating game variables and information in every turn, such as updating money upkeep, monitoring global crisis, monitoring crisis resolve progress, checking global index updates, updating other cities index after a trade, and checking if the game has ended

```
public void NextTurn()
{

    UpdatePlayerMoney(turnMoney);
    continent_AI.updateContinentIndexAfterTrade();

    emergencyScript.solvingCrisis();
    emergencyScript.emergencyUpdate();
    CheckPlanning();
    turnMoneyText.text = turnMoney.ToString() + " per turn";
    turn++;
    turnMsg = "Turn: ";
    gameTurnText.text = turnMsg + turn.ToString() + " /30";

    if(turn >= 30)
    {
        ending_Sequences.finalTurn();
    }

    if(nextRandomEventOccurance == turn)
    {
        currentRandomEventTurn = nextRandomEventOccurance + 5;
        EmergencyEventCaller();
    }

    if (currentRandomEventTurn == turn)
    {
        EmergencyEventEnder();
    }

    if (everyFiveTurn == turn)
    {
        globalScript.globalIndexEffects();
        everyFiveTurn = turn + 5;
    }

}
```

(Figure 7.2 Game_manager script)

- Index level
    - For updating all 3 local index levels to the interface's index bar

```
private void updateIndex(int econLV, int envLV, int socLV)
{
    float newRatioECON = (econLV / 12.0f) + ratioECON;
    float newRatioENV = (envLV / 12.0f) + ratioENV;
    float newRatioSOC = (socLV / 12.0f) + ratioSOC;

    progressBarECON.newIndexUpdate(newRatioECON);
    progressBarENV.newIndexUpdate(newRatioENV);
    progressBarSOC.newIndexUpdate(newRatioSOC);

    ratioECON = newRatioECON;
    ratioENV = newRatioENV;
    ratioSOC = newRatioSOC;
}
```

(Figure 7.3 Game_manager script)

- Income system
    - For updating and displaying player's banking status, as well as their income per turn

```
public void UpdatePlayerMoney(int changes)
{
    if(changes < 0 && playerMoney < -(changes))
    {
        playerMoney = 0;
    }
    else
    {
        playerMoney += changes;
    }

    playerMoneyText.text = playerMoney.ToString();

}
```

(Figure 7.4 Game_manager script)

- Global Crisis caller and ender
    - For calling and ending global crisis based on game's number of turns

```
public void EmergencyEventCaller()
{
    randomEventWarning.SetActive(true);
    emergencyScript.emergencyEventInitialise();
    nextRandomEventOccurance = turn + 5 + Random.Range(4, 6);
}

public void EmergencyEventEnder()
{
    randomEventWarning.SetActive(false);
}
```

(Figure 7.5 Game_manager script)

- Building demolish update
    - When called by Tile_placer, this function will perform necessary updates of reversing the effect on index level and production cost

```
public void DemolishBuilding(GameObject buidling, int tileNum)
{
    avaliableWaitingSlot[tileNum] = true;
    occupiedTile[tileNum] = false;
    turnMoney -= buildingMoneyReturn[tileNum];
    updateIndex(-waitingLV[0, tileNum], -waitingLV[1, tileNum], -waitingLV[2, tileNum]);
    Destroy(buidling);
}
```

(Figure 7.6 Game_manager script)

- Display Crisis News Update
    - When called by Global_Crisis, this function will display a crisis update news panel, the description varies according to the type of message call. Player can also close the panel after viewing.

```
public void displayCrisisNews(int typeOfMsg)
{
    switch (typeOfMsg)
    {
        case 0:
            crisisNewsText.text = "Begin resolve project, need to wait for 3 turns!";
            break;
        case 1:
            crisisNewsText.text = "No tile are related to resolve, start building one!";
            break;
        case 2:
            crisisNewsText.text = "Crisis evaded, no penalty! Good job!";
            break;
        case 3:
            crisisNewsText.text = "Whoops... a crisis has occured, global index decreased!";
            break;
        default:
            break;
    }
    crisisNews.SetActive(true);
}

public void closeCrisisNews()
{
    crisisNews.SetActive(false);
}
```

(Figure 7.7 Game_manager script)

Tile_palcer

Script for mouse input for tile construction and tile purchase, and perform tasks for tile placement indications

- Tile state update
    - Responsible for visualisation of tile selection, buildings and card deck animation
    - Display indicator when player's mouse hover on tiles
    - Remains if player's mouse clicked the owned tiles
    - Disappear when player's mouse leaves the tile area of selection / choose a building to be built on
    - Display tile purchase panel if player's mouse clicked the non-purchased tiles
    - Display building demolish panel if player's mouse clicked the occupied tiles

```
// Update is called once per frame
void Update()
{
    if (tileSelected)
    {

        Vector2 positionTCardDeck = TCardDeck.anchoredPosition;
        Vector2 targetTCardDeck = new Vector2(originalTCardDeck.x, openedTCardDeck);

        TCardDeck.anchoredPosition = Vector2.SmoothDamp(positionTCardDeck, targetTCardDeck, ref velocityTCardDeck, HorizontalSpeed * Time.deltaTime);

    }
    else
    {

        Vector2 positionTCardDeck = TCardDeck.anchoredPosition;
        Vector2 targetTCardDeck = originalTCardDeck;

        TCardDeck.anchoredPosition = Vector2.SmoothDamp(positionTCardDeck, targetTCardDeck, ref velocityTCardDeck, HorizontalSpeed * Time.deltaTime);

    }

    if (tileSelected && Input.GetKeyDown(KeyCode.Escape)) //cancel tile select
        CancelTileSelection();
    if (!tileSelected && Time.time - lastUpdateTime > placementIndicatorUpdateRate) //update crusor position
    {
        lastUpdateTime = Time.time;
        curPlacementPos = tile_selector.inst.GetCurTilePosition();
        placementIndicator.transform.position = curPlacementPos;
    }
    if (!currentlyPlacing && !currentlyBuying) //toggle placer when crusor land on a tile, if it's not placing any building
    {
        CheckTile();
    }
```

(Figure 8.1. Tile_placer script)

```
    }
    if (!currentlyPlacing && !currentlyBuying) //toggle placer when crusor land on a tile, if it's not placing any building
    {
        CheckTile();
    }
    if (currentlyPlacing && Input.GetMouseButtonDown(0)) //selecting a building to place
    {
        if (occupiedTile[selectedTileNum] == true) //if tile has already been occupied
        {

            demolishBuildingPannel.SetActive(true);
            placementIndicator.transform.position = ownedTiles[selectedTileNum].transform.position;
        }
        else
        {

            selectionCancelButton.SetActive(true);
            tileSelected = true;
            placementIndicator.transform.position = ownedTiles[selectedTileNum].transform.position;
        }
    }
    if (currentlyBuying && Input.GetMouseButtonDown(0)) //selecting a tile to buy
    {
        tileBought = true;
        buyTilePannel.SetActive(true);
        tileCostMessage.text = "Are you sure you want to buy this tile?\nIt will cost you $" + buyTileCost.ToString() + ".";
    }
    if ((currentlyPlacing && !tileSelected) || (currentlyBuying && !tileBought)) //check if mouse is still hovering on tile
    {
        CheckBuildingPlacement();
    }
}
```

(Figure 8.2. Tile_placer script)

```
public void CheckTile()
{
    for (int n = 0; n < ownedTiles.Count; n++)
    {

        if (ownedTiles[n].transform.position.x - 10.0f < curPlacementPos.x && ownedTiles[n].transform.position.x + 10.0f > curPlacementPos.x
            && ownedTiles[n].transform.position.z - 10.0f < curPlacementPos.z && ownedTiles[n].transform.position.z + 10.0f > curPlacementPos.z)
        {
            currentlyPlacing = true;
            placementIndicator.SetActive(true);


            selectedTileNum = n;
        }
    }
    for (int n = 0; n < avaliableTiles.Count; n++)
    {

        if (avaliableTiles[n].transform.position.x - 10.0f < curPlacementPos.x && avaliableTiles[n].transform.position.x + 10.0f > curPlacementPos.x
            && avaliableTiles[n].transform.position.z - 10.0f < curPlacementPos.z && avaliableTiles[n].transform.position.z + 10.0f > curPlacementPos.z)
        {

            currentlyBuying = true;
            tile_material[ownedTiles.Count+n].color = avaliableSelectedColor;


            buyTileNum = n;
        }
    }
}
```

(Figure 8.3. Tile_placer script)

```
public void CheckBuildingPlacement()
{
    if (selectedTileNum != -1)
    {
        if (!(ownedTiles[selectedTileNum].transform.position.x - 10.0f < curPlacementPos.x && ownedTiles[selectedTileNum].transform.position.x + 10.0f > curPlacementPos.x
            && ownedTiles[selectedTileNum].transform.position.z - 10.0f < curPlacementPos.z && ownedTiles[selectedTileNum].transform.position.z + 10.0f > curPlacementPos.z))
        {

            CancelBuildingPlacement();
            selectedTileNum = -1;
        }
    }
    if (buyTileNum != -1)
    {
        if (!(avaliableTiles[buyTileNum].transform.position.x - 10.0f < curPlacementPos.x && avaliableTiles[buyTileNum].transform.position.x + 10.0f > curPlacementPos.x
            && avaliableTiles[buyTileNum].transform.position.z - 10.0f < curPlacementPos.z && avaliableTiles[buyTileNum].transform.position.z + 10.0f > curPlacementPos.z))
        {
            CancelBuyTile();
        }
    }
}

public void CancelTileSelection()
{
    tileSelected = false;
    selectionCancelButton.SetActive(false);
    placementIndicator.SetActive(false);
}

public void CancelBuildingPlacement()
{
    currentlyPlacing = false;
    selectionCancelButton.SetActive(false);
    placementIndicator.SetActive(false);
}
```

(Figure 8.4. Tile_placer script)

- Buying tiles
    - Perform updates after the purchase such as money reduction from Game_manager and calling tile texture updates

```
public void BuyTile()
{
    if (game_Manager.playerMoney < buyTileCost)
    {
        tileNoMoneyWarning.SetActive(true);
    }
    else
    {
        ownedTiles.Add(avaliableTiles[buyTileNum]);

        avaliableTiles.Remove(avaliableTiles[buyTileNum]);
        game_Manager.UpdatePlayerMoney(-buyTileCost);

        buyTileCost += buyTileCostConstant;

        tileBought = false;

        buyTileNum = -1;
        currentlyBuying = false;
        buyTilePannel.SetActive(false);
        UpdateTileTextures();
    }

}
public void CancelBuyTile()
{
    tileNoMoneyWarning.SetActive(false);
    tileBought = false;

    tile_material[ownedTiles.Count + buyTileNum].color = avaliableIdleColor;
    buyTileNum = -1;
    currentlyBuying = false;
    buyTilePannel.SetActive(false);
}
```

(Figure 8.5. Tile_placer script)

- Update Tile Textures
    - For updating tile textures after the non-purchased tile is being bought by player

```
private void UpdateTileTextures()
{
    int numOfTiles = 0;
    for (int n = 0; n < ownedTiles.Count; n++)
    {

        tile_material[numOfTiles] = ownedTiles[n].GetComponent<Renderer>().material;
        tile_material[numOfTiles].color = ownedColor;
        numOfTiles++;
    }
    for (int n = 0; n < avaliableTiles.Count; n++)
    {

        tile_material[numOfTiles] = avaliableTiles[n].GetComponent<Renderer>().material;
        tile_material[numOfTiles].color = avaliableIdleColor;
        numOfTiles++;
    }
}
```

(Figure 8.6. Tile_placer script)

- Demolishing buildings
    - Perform updates after the player decided whether to demolish buildings or not, the demolish operation continues in main script game_manager.

```
public void DemolishBuilding()
{
    game_Manager.DemolishBuilding(occupiedTileWithBuildings[buyTileNum], selectedTileNum);

    CancelDemolish();

}
public void CancelDemolish()
{
    selectedTileNum = -1;
    currentlyPlacing = false;
    demolishBuildingPannel.SetActive(false);
}
```

(Figure 8.7. Tile_placer script)

Game_camera_controller

Script for camera controls, perform tasks for camera movements

- Movement control
    - Responsible for camera position and rotation movements

```
void Update()
{


    if (Input.GetMouseButton(1))
    {
        float x = Input.GetAxis("Mouse X");
        float y = Input.GetAxis("Mouse Y");
        curXRot += x * rotateSpeed;
        curXRot = Mathf.Clamp(curXRot, minXRot, maxXRot);

        curYRot += -y * rotateSpeed;
        curYRot = Mathf.Clamp(curYRot, minYRot, maxYRot);
        transform.eulerAngles = new Vector3(curYRot, curXRot, 0.0f);
    }


    Vector3 forward = cam.transform.forward;
    forward.y = 0.0f;
    forward.Normalize();
    Vector3 right = cam.transform.right.normalized;
    float moveX = Input.GetAxisRaw("Horizontal");
    float moveZ = Input.GetAxisRaw("Vertical");
    Vector3 dir = forward * moveZ + right * moveX;
    dir.Normalize();
    dir *= moveSpeed * Time.deltaTime;

    cam.transform.position += dir;
}
```

(Figure 9. game_camera_controller script)

Tile_selector

Script for tile hover inspection, perform tasks for script Tile_placer

- Mouse to object indicator
    - Find if the mouse is pointing on a tile or not by determining if its cursor position in-game and in-game objects are in the same place

```
public Vector3 GetCurTilePosition()
{
    if (IsPointerOverUIObject()) //poiniting at ui
        return new Vector3(0, -99, 0);
    Plane plane = new Plane(Vector3.up, Vector3.zero);
    Ray ray = cam.ScreenPointToRay(Input.mousePosition);
    float rayOut = 0.0f;
    if (plane.Raycast(cam.ScreenPointToRay(Input.mousePosition), out rayOut))
    {
        Vector3 newPos = ray.GetPoint(rayOut) - new Vector3(0.5f, 0.0f, 0.5f);
        //Debug.Log("yo the tile you picked is at" + newPos);
        return new Vector3(Mathf.CeilToInt(newPos.x), 0, Mathf.CeilToInt(newPos.z));
    }
    return new Vector3(0, -99, 0);
}

public static bool IsPointerOverUIObject()
{
    PointerEventData eventDataCurrentPosition = new PointerEventData(EventSystem.current);
    eventDataCurrentPosition.position = new Vector2(Input.mousePosition.x, Input.mousePosition.y);
    List<RaycastResult> results = new List<RaycastResult>();
    EventSystem.current.RaycastAll(eventDataCurrentPosition, results);
    return results.Count > 0;
}
```

(Figure 10. Tile_selector script)

EndingScene

Script for displaying the ending sequence of the game

- Display ending pages
    - Section 1 of ending shows how well player's city has performed (i.e. local)
    - Section 2 of ending shows how well the world's cities has performed (i.e. global)
    - Section 3 of ending display the final score of the player, and take input for the scoreboard (i.e. score)

```
// Update is called once per frame
void Update()
{
    if (isPageLocal)
    {
        endingPage1.SetActive(true);
        endingPage2.SetActive(false);
        endingPage3.SetActive(false);
        if (mainScript.ratioECON <= 1.0f / 3.0f)
        {
            endEconText.text = "Your city's unemployment rate is sky high... (" + mainScript.ratioECON.ToString() + " / 1)";
        }
        else if (mainScript.ratioECON > 1.0f / 3.0f && mainScript.ratioECON <= 2.0f / 3.0f)
        {
            endEconText.text = "Your city's economy is doing a-okay, no big contributions, but no disasters either... (" + mainScript.ratioECON.ToString() + " / 1)";
        }
        else
        {
            endEconText.text = "Your city's economy is booming... (" + mainScript.ratioECON.ToString() + " / 1)";
        }

        if (mainScript.ratioENV <= 1.0f / 3.0f)
        {
            endEnvText.text = "Your city suffers from all kinds of polution... (" + mainScript.ratioENV.ToString() + " / 1)";
        }
        else if (mainScript.ratioENV > 1.0f / 3.0f && mainScript.ratioENV <= 2.0f / 3.0f)
        {
            endEnvText.text = "Your city's enviornment is at the very least acceptable, pollutions are still a problem in your city though... (" + mainScript.ratioENV.ToString() + " / 1)";
        }
        else
        {
            endEnvText.text = "Your city's enviroment is flourishing... (" + mainScript.ratioENV.ToString() + " / 1)";
        }

        if (mainScript.ratioSOC <= 1.0f / 3.0f)
        {
            endSocText.text = "Your city is rotting with crime and suffering... (" + mainScript.ratioSOC.ToString() + " / 1)";
        }
        else if (mainScript.ratioSOC > 1.0f / 3.0f && mainScript.ratioSOC <= 2.0f / 3.0f)
        {
            endSocText.text = "Your citizens are not very happy about your city developement, but at least they're not rioting... (" + mainScript.ratioSOC.ToString() + " / 1)";
        }
        else
        {
            endSocText.text = "Your city's citizens are very satisfied with their life... (" + mainScript.ratioSOC.ToString() + " / 1)";
        }
    }
```

(Figure 11.1 EndingScene script)

```
    }
    else if(isPageGlobal)
    {
        endingPage1.SetActive(false);
        endingPage2.SetActive(true);
        endingPage3.SetActive(false);

        if (globalScript.globalRatioECON <= 1.0f / 3.0f)
        {
            endGlobEconText.text = "Global economic depression is here... (" + globalScript.globalRatioECON.ToString() + " / 1)";
        }
        else if (globalScript.globalRatioECON > 1.0f / 3.0f && globalScript.globalRatioECON <= 2.0f / 3.0f)
        {
            endGlobEconText.text = "The world's economy is in stagnation... (" + globalScript.globalRatioECON.ToString() + " / 1)";
        }
        else
        {
            endGlobEconText.text = "The world's economy is flourishing with great future... (" + globalScript.globalRatioECON.ToString() + " / 1)";
        }

        if (globalScript.globalRatioENV <= 1.0f / 3.0f)
        {
            endGlobEnvText.text = "Global warming is very severe, extreme climates are all around the world... (" + globalScript.globalRatioENV.ToString() + " / 1)";
        }
        else if (globalScript.globalRatioENV > 1.0f / 3.0f && globalScript.globalRatioENV <= 2.0f / 3.0f)
        {
            endGlobEnvText.text = "Global warming is an inconvenience to some people, not great, but not terrible... (" + globalScript.globalRatioENV.ToString() + " / 1)";
        }
        else
        {
            endGlobEnvText.text = "Global environment has been greatly improved, the earth is healing... (" + globalScript.globalRatioENV.ToString() + " / 1)";
        }

        if (globalScript.globalRatioSOC <= 1.0f / 3.0f)
        {
            endGlobSocText.text = "Riots and civil wars are happening in every corner on earth, our civil rights are at risk... (" + globalScript.globalRatioSOC.ToString() + " / 1)";
        }
        else if (globalScript.globalRatioSOC > 1.0f / 3.0f && globalScript.globalRatioSOC <= 2.0f / 3.0f)
        {
            endGlobSocText.text = "Civil right movements are a common thing around the world, but the world doesn't seem to care that much... (" + globalScript.globalRatioSOC.ToString() + " / 1)";
        }
        else
        {
            endGlobSocText.text = "The world's enjoying its long deserved peace, generations are living happily ever after... (" + globalScript.globalRatioSOC.ToString() + " / 1)";
        }

    }
    else //isPageScores
    {
        endingPage1.SetActive(false);
        endingPage2.SetActive(false);
        endingPage3.SetActive(true);
        highScore();
    }
```

(Figure 11.2 EndingScene script)

- End score and scoreboard
    - Responsible for displaying the final game score of the player, as well as determining name input exist for saving in scoreboard

```
public void highScore()
{
    resultScore = (mainScript.ratioECON + mainScript.ratioENV + mainScript.ratioSOC + globalScript.globalRatioECON + globalScript.globalRatioENV + globalScript.globalRatioSOC)* 16.6666666667f;

    endScoreText.text = resultScore.ToString("F2");
}

public void closeWarningOnEdit()
{
    scoreInputWarning.SetActive(false);
}

public void finalTurn()
{
    mainScript.endingDisable.SetActive(false);
    mainScript.endingEnable.SetActive(true);
}
```

(Figure 11.3 EndingScene script)

- Button decisions from ending pages
    - Responsible for button actions in all ending pages, such as switching pages and calling scoreboard function to save high score for scoreboard.

```
public void goToGlobal()
{
    if (isPageLocal)
    {
        localButton.SetActive(true);
        globalButton.SetActive(true);
        isPageLocal = false;
        isPageGlobal = true;
        isPageScore = false;
        globalOrMenu.text = "Scores";
    }
    else
    {
        if (isPageGlobal)
        {
            localButton.SetActive(true);
            globalButton.SetActive(true);
            isPageLocal = false;
            isPageGlobal = false;
            isPageScore = true;
            globalOrMenu.text = "Submit";
        }
        else
        {
            if(scoreName.text.Length <= 3 && scoreName.text.Length > 0)
            {
                isInputNotEmpty = true;
            }
            if (isInputNotEmpty)
            {
                SendScoreFunction(scoreName.text, resultScore);
                print("scoreName is: " + scoreName.text);
            }
            else
            {
                scoreInputWarning.SetActive(true);
            }
        }
    }
}
```

(Figure 11.4 EndingScene script)

```
public void goToLocal()
{
    if (isPageGlobal)
    {

        localButton.SetActive(false);
        globalButton.SetActive(true);
        isPageLocal = true;
        isPageGlobal = false;
        isPageScore = false;
        globalOrMenu.text = "Global";
    }
    else //isPageScore
    {
        localButton.SetActive(true);
        globalButton.SetActive(true);
        isPageLocal = false;
        isPageGlobal = true;
        isPageScore = false;
        globalOrMenu.text = "Scores";
    }
}
```

(Figure 11.5 EndingScene script)

- Send Score Function
    - Responsible for saving score results and player name into the leader-board by JSON.

```
public void SendScoreFunction(string name, float value)
{

    if (name == null || string.IsNullOrEmpty(name.Trim()))
    {
        name = "Sheep";
    }

    if (name.Length > 30)
    {
        name = name.Substring(0, 30);
    }

    Hashtable data = new Hashtable();
    data.Add("name", name);
    data.Add("value", value);

    UnityHTTP.Request theRequest = new UnityHTTP.Request("post", "https://fyp-gamedatabase-default-rtdb.firebaseio.com/scores.json", data);
    theRequest.Send((request) => {
        Hashtable jsonObj = (Hashtable)JSON.JsonDecode(request.response.Text);
        if (jsonObj == null)
        {
            Debug.LogError("server returned null or malformed response ):");
        }
        TopScore.SetPlayer((string)jsonObj["name"], name, value);

    });
}
```

(Figure 11.6 EndingScene script)

GlobalManager

Script for controlling global panel of the game, perform tasks for global interface and global in-game objects

- Interface updates

- Perform animations for global interface objects according to the control states (opening / closing trading panel, opening / closing city panel)

```
void Update()
{
    if (continentIsOn)
    {
        if (tradeIsOn)
        {
            Vector2 positionCard = continentCard.anchoredPosition;
            Vector2 targetCard = new Vector2(openedContinentCards - 380, 0);

            continentCard.anchoredPosition = Vector2.SmoothDamp(positionCard, targetCard, ref velocityCard, HorizontalSpeed * Time.deltaTime);

            Vector2 positionTradeCard = TradeCards.anchoredPosition;
            Vector2 targetTradeCard = new Vector2(originalTradeCards.x, openedTradeCards);

            TradeCards.anchoredPosition = Vector2.SmoothDamp(positionTradeCard, targetTradeCard, ref velocityTradeCards, HorizontalSpeed * Time.deltaTime);

            Vector2 positionTCardDeck = TCardDeck.anchoredPosition;
            Vector2 targetTCardDeck = new Vector2(originalTCardDeck.x, openedTCardDeck);

            TCardDeck.anchoredPosition = Vector2.SmoothDamp(positionTCardDeck, targetTCardDeck, ref velocityTCardDeck, HorizontalSpeed * Time.deltaTime);

        }
        else
        {
            Vector2 positionLand = rectTransform.anchoredPosition;

            Vector2 targetLand = new Vector2(openedContinentLands, 0);

            rectTransform.anchoredPosition = Vector2.SmoothDamp(positionLand, targetLand, ref velocityContinent, HorizontalSpeed * Time.deltaTime);


            Vector2 positionCard = continentCard.anchoredPosition;
            Vector2 targetCard = new Vector2(openedContinentCards, 0);

            continentCard.anchoredPosition = Vector2.SmoothDamp(positionCard, targetCard, ref velocityCard, HorizontalSpeed * Time.deltaTime);

            Vector2 positionTradeCard = TradeCards.anchoredPosition;
            Vector2 targetTradeCard = originalTradeCards;

            TradeCards.anchoredPosition = Vector2.SmoothDamp(positionTradeCard, targetTradeCard, ref velocityTradeCards, HorizontalSpeed * Time.deltaTime);

            Vector2 positionTCardDeck = TCardDeck.anchoredPosition;
            Vector2 targetTCardDeck = originalTCardDeck;

            TCardDeck.anchoredPosition = Vector2.SmoothDamp(positionTCardDeck, targetTCardDeck, ref velocityTCardDeck, HorizontalSpeed * Time.deltaTime);

        }

    }
    else{
        Vector2 positionLand = rectTransform.anchoredPosition;
        Vector2 targetLand = originalContinent;

        rectTransform.anchoredPosition = Vector2.SmoothDamp(positionLand, targetLand, ref velocityContinent, HorizontalSpeed * Time.deltaTime);


        Vector2 positionCard = continentCard.anchoredPosition;
        Vector2 targetCard = originalCard;

        continentCard.anchoredPosition = Vector2.SmoothDamp(positionCard, targetCard, ref velocityCard, HorizontalSpeed * Time.deltaTime);
    }
}
```

(Figure 12.1 GlobalManager script)

- OnClickedContinent
    - Responsible for displaying the right city (North America, South America, Africa, Asia, Europe, Oceania) according to player's mouse input during global panel, and choosing cities for players to start during game initialisation, while also determines if the chosen city is player's own city to avoid displaying their own city

```
public void OnClickedContinent()
{
    continentIsOn = true;
    string continentName = EventSystem.current.currentSelectedGameObject.name;
    print("continentNumPicked: " + continentNumPicked);
    switch (continentName)
    {
    case "SAmerica":
            if (gameInitilization)
            {
                continentNumPicked = 0;
                tmpConImage.GetComponent<Image>().sprite = continentInfo[0].continentImage;
                continentScript.displayIndex(continentInfo[0]);
            }
            else
            {
                if(continentNumPicked != 0)
                {
                    tmpConImage.GetComponent<Image>().sprite = continentInfo[0].continentImage;
                    continentScript.displayIndex(continentInfo[0]);
                }
                else
                {
                    continentIsOn = false;
                }
            }
            break;
    case "NAmerica":
            if (gameInitilization)
            {
                continentNumPicked = 1;
                tmpConImage.GetComponent<Image>().sprite = continentInfo[1].continentImage;
                continentScript.displayIndex(continentInfo[1]);
            }
            else
            {
                if (continentNumPicked != 1)
                {
                    tmpConImage.GetComponent<Image>().sprite = continentInfo[1].continentImage;
                    continentScript.displayIndex(continentInfo[1]);
                }
```

(Figure 12.2 GlobalManager script)

```
                }
                else
                {
                    continentIsOn = false;
                }
            }
            break;
    case "oceania":
            if (gameInitilization)
            {
                continentNumPicked = 2;
                tmpConImage.GetComponent<Image>().sprite = continentInfo[2].continentImage;
                continentScript.displayIndex(continentInfo[2]);
            }
            else
            {
                if (continentNumPicked != 2)
                {
                    tmpConImage.GetComponent<Image>().sprite = continentInfo[2].continentImage;
                    continentScript.displayIndex(continentInfo[2]);
                }
                else
                {
                    continentIsOn = false;
                }
            }
            break;
    case "asia":
            if (gameInitilization)
            {
                continentNumPicked = 3;
                tmpConImage.GetComponent<Image>().sprite = continentInfo[3].continentImage;
                continentScript.displayIndex(continentInfo[3]);
            }
            else
            {
                if (continentNumPicked != 3)
                {
                    tmpConImage.GetComponent<Image>().sprite = continentInfo[3].continentImage;
                    continentScript.displayIndex(continentInfo[3]);
                }
                else
```

(Figure 12.3 GlobalManager script)

```
                else
                {
                    continentIsOn = false;
                }
            }
            break;
        case "europe":
            if (gameInitilization)
            {
                continentNumPicked = 4;
                tmpConImage.GetComponent<Image>().sprite = continentInfo[4].continentImage;
                continentScript.displayIndex(continentInfo[4]);
            }
            else
            {
                if (continentNumPicked != 4)
                {
                    tmpConImage.GetComponent<Image>().sprite = continentInfo[4].continentImage;
                    continentScript.displayIndex(continentInfo[4]);
                }
                else
                {
                    continentIsOn = false;
                }
            }
            break;
        case "africa":
            if (gameInitilization)
            {
                continentNumPicked = 5;
                tmpConImage.GetComponent<Image>().sprite = continentInfo[5].continentImage;
                continentScript.displayIndex(continentInfo[5]);
            }
            else
            {
                if (continentNumPicked != 5)
                {
                    tmpConImage.GetComponent<Image>().sprite = continentInfo[5].continentImage;
                    continentScript.displayIndex(continentInfo[5]);
                }
                else
                {
                    continentIsOn = false;
                }
            }
            break;
        default:
            break;
    }
}
```

(Figure 12.4 GlobalManager script)

- Button Interactions
    - Responsible for switching trade panel and city panel, while also responsible for initialising starting city's information such as recording index level and its city's name, as well as calculating the new global index levels

```
public void OnClickedCardsQuit()
{
    if (tradeIsOn)
    {
        tradeIsOn = false;
        cardTradeText.SetActive(false);
        cardTradeButton.SetActive(true);
    }
    else
    {
        continentIsOn = false;
        if (gameInitilization)
        {
            continentNumPicked = -1;
        }
    }
}

public void OnClickedCardsTrade()
{
    if (!gameInitilization)
    {
        tradeIsOn = true;
        cardTradeText.SetActive(true);
        cardTradeButton.SetActive(false);
    }
    else
    {
        startContinentButton.text = "Trade";
        mainScript.updateIndex((int)(continentInfo[continentNumPicked].econRatio * 12.0f) - 6, (int)(continentInfo[continentNumPicked].envRatio * 12.0f) - 6, (int)(continentInfo[continentNumPicked].socRatio * 12.0f) - 6);

        float tmpGlobalRatioECON = 0.0f;
        float tmpGlobalRatioENV = 0.0f;
        float tmpGlobalRatioSOC = 0.0f;
        for (int num = 0; num < continentInfo.Length; num++)
        {
            if (continentNumPicked == num)
            {
                continue;
            }
            tmpGlobalRatioECON += continentInfo[num].econRatio;
            tmpGlobalRatioENV += continentInfo[num].envRatio;
            tmpGlobalRatioSOC += continentInfo[num].socRatio;
        }
        globalRatioECON = tmpGlobalRatioECON / continentInfo.Length;
        globalRatioENV = tmpGlobalRatioENV / continentInfo.Length;
        globalRatioSOC = tmpGlobalRatioSOC / continentInfo.Length;

        gameInitilization = false;
        returnButton.SetActive(true);
        globalUI.SetActive(false);
        startDescription.SetActive(false);
        continentIsOn = false;
        print("continentNumPicked: " + continentNumPicked);
    }
}
```

(Figure 12.5 GlobalManager script)

- updateContinentPrefab
    - Responsible for updating cities (i.e. continents) information (mainly the index level) after a trade is successful

```
public void updateContinentPrefab(ContinentScript newContinentUpdate)
{
    string continentName = newContinentUpdate.continentName;
    switch (continentName)
    {
        case "SAmerica":
            continentInfo[0] = newContinentUpdate;
            break;
        case "NAmerica":
            continentInfo[1] = newContinentUpdate;
            break;
        case "oceania":
            continentInfo[2] = newContinentUpdate;
            break;
        case "asia":
            continentInfo[3] = newContinentUpdate;
            break;
        case "europe":
            continentInfo[4] = newContinentUpdate;
            break;
        case "africa":
            continentInfo[5] = newContinentUpdate;
            break;
        default:
            break;
    }
}
```

(Figure 12.6 GlobalManager script)

- Global Index level update
    - Similar to the one on game_manager, these functions are responsible for updating all 3 global index levels, the function with input is for crisis event index level updates, the function with no parameters is for automatic updates in each turn (from updates like trading)

```
public void updateGlobalIndex(int econLV, int envLV, int socLV)
{
    float tmpGlobalRatioECON = 0.0f;
    float tmpGlobalRatioENV = 0.0f;
    float tmpGlobalRatioSOC = 0.0f;
    for (int num = 0; num< continentInfo.Length; num++)
    {
        if (continentNumPicked == num)
        {
            continue;
        }
        tmpGlobalRatioECON += continentInfo[num].econRatio;
        tmpGlobalRatioENV += continentInfo[num].envRatio;
        tmpGlobalRatioSOC += continentInfo[num].socRatio;
    }
    globalRatioECON = tmpGlobalRatioECON / continentInfo.Length;
    globalRatioENV = tmpGlobalRatioENV / continentInfo.Length;
    globalRatioSOC = tmpGlobalRatioSOC / continentInfo.Length;

    float newRatioECON = (econLV / 12.0f) + globalRatioECON;
    float newRatioENV = (envLV / 12.0f) + globalRatioENV;
    float newRatioSOC = (socLV / 12.0f) + globalRatioSOC;


    globalRatioECON = newRatioECON;
    globalRatioENV = newRatioENV;
    globalRatioSOC = newRatioSOC;
}

public void updateGlobalIndex()
{
    float tmpGlobalRatioECON = 0.0f;
    float tmpGlobalRatioENV = 0.0f;
    float tmpGlobalRatioSOC = 0.0f;
    for (int num = 0; num < continentInfo.Length; num++)
    {
        if (continentNumPicked == num)
        {
            continue;
        }
        tmpGlobalRatioECON += continentInfo[num].econRatio;
        tmpGlobalRatioENV += continentInfo[num].envRatio;
        tmpGlobalRatioSOC += continentInfo[num].socRatio;
    }
    globalRatioECON = tmpGlobalRatioECON / continentInfo.Length;
    globalRatioENV = tmpGlobalRatioENV / continentInfo.Length;
    globalRatioSOC = tmpGlobalRatioSOC / continentInfo.Length;
}
```

(Figure 12.7 GlobalManager script)

- Global Index impacts on local index
    - Based on how well the global index is, this function affects and updates the local index every 5 turns

```
public void globalIndexEffects()
{
    int newIndexEconLV;
    int newIndexEnvLV;
    int newIndexSocLV;
    if (globalRatioECON <= 1.0f / 3.0f)
    {
        newIndexEconLV = -1;
    }
    else if(globalRatioECON > 1.0f / 3.0f && globalRatioECON <= 2.0f / 3.0f)
    {
        newIndexEconLV = 0;
    }
    else
    {
        newIndexEconLV = 1;
    }
    if (globalRatioENV <= 1.0f / 3.0f)
    {
        newIndexEnvLV = -1;
    }
    else if (globalRatioENV > 1.0f / 3.0f && globalRatioENV <= 2.0f / 3.0f)
    {
        newIndexEnvLV = 0;
    }
    else
    {
        newIndexEnvLV = 1;
    }
    if (globalRatioSOC <= 1.0f / 3.0f)
    {
        newIndexSocLV = -1;
    }
    else if (globalRatioSOC > 1.0f / 3.0f && globalRatioSOC <= 2.0f / 3.0f)
    {
        newIndexSocLV = 0;
    }
    else
    {
        newIndexSocLV = 1;
    }

    mainScript.updateIndex(newIndexEconLV, newIndexEnvLV, newIndexSocLV);
}
```

(Figure 12.8 GlobalManager script)

Continent_AI

Script for displaying and updating global cities' information, and controlling their behaviours

- Display cities index level
    - Responsible for displaying city index level on interface when the city is chosen

```
public void displayIndex(ContinentScript continentPrefab)
{
    currentContinent = continentPrefab;
    var newRatioIndex = new float[3] { 0f, 0f, 0f };
    newRatioIndex[0] = continentPrefab.econRatio;
    newRatioIndex[1] = continentPrefab.envRatio;
    newRatioIndex[2] = continentPrefab.socRatio;

    for (int n = 0; n < 3; n++)
    {
        if (newRatioIndex[n] < 0.5)
        {
            float newRatio = newRatioIndex[n] * 2;
            indexStatusColor[n].color = new Color(1.0f, newRatio, 0.0f);
        }
        if (newRatioIndex[n] >= 0.5)
        {
            float newRatio = newRatioIndex[n] * 2 - 1;
            indexStatusColor[n].color = new Color(1 - newRatio, 1.0f, 0.0f);
        }
    }

}
```

(Figure 13.1 Continent_AI script)

- Update cities index level
    - Responsible for updating city index level after a successful trade operation

```
public void updateIndex(int econLV, int envLV, int socLV, ContinentScript continentPrefab)
{
    currentContinent = continentPrefab;
    var newRatioIndex = new float[3] { 0f, 0f, 0f };
    newRatioIndex[0] = (econLV / 12.0f) + continentPrefab.econRatio;
    newRatioIndex[1] = (envLV / 12.0f) + continentPrefab.envRatio;
    newRatioIndex[2] = (socLV / 12.0f) + continentPrefab.socRatio;

    currentContinent.econRatio = newRatioIndex[0];
    currentContinent.envRatio = newRatioIndex[1];
    currentContinent.socRatio = newRatioIndex[2];

    displayIndex(currentContinent);
    globalManager.updateContinentPrefab(currentContinent);
}
```

(Figure 13.2 Continent_AI script)

- City’s behaviour on trading operation
    - Responsible for game decisions during a trade operation (the complete decision flow is on section “Game AI”)

```
public void TradingDecision(TileScript buildingPreset)
{
    waitingBuildingPreset = buildingPreset;

    var preferenceRatio = new float[3] { 0f, 0f, 0f };

    print("card preset info, econLV: " + waitingBuildingPreset.econLV + "envLV: " + waitingBuildingPreset.envLV + "socLV: " + waitingBuildingPreset.socLV);

    if ( 0.5 > currentContinent.econRatio && waitingBuildingPreset.econLV > 0)
    {
        preferenceRatio[0] = currentContinent.tradePreference[0] * (float)waitingBuildingPreset.econLV;
    }
    else
    {
        preferenceRatio[0] = 0;
    }

    if (0.5 > currentContinent.envRatio && waitingBuildingPreset.envLV > 0)
    {
        preferenceRatio[1] = currentContinent.tradePreference[1] * (float)waitingBuildingPreset.envLV;
    }
    else
    {
        preferenceRatio[1] = 0;
    }

    if (0.5 > currentContinent.socRatio && waitingBuildingPreset.socLV > 0)
    {
        preferenceRatio[2] = currentContinent.tradePreference[2] * (float)waitingBuildingPreset.socLV;
    }
    else
    {
        preferenceRatio[2] = 0;
    }

    float preferenceResult = preferenceRatio[0] + preferenceRatio[1] + preferenceRatio[2];

    if (preferenceResult > 0)
    {
        if (!isTradeCompleteInThisTurn)
        {

            print("trade successful!  trade preference: " + preferenceResult);
            isTradeCompleteInThisTurn = true;
            completedTradeIndexUpdate[0] = waitingBuildingPreset.econLV;
            completedTradeIndexUpdate[1] = waitingBuildingPreset.envLV;
            completedTradeIndexUpdate[2] = waitingBuildingPreset.socLV;
            completedTradeWithThisContinent = currentContinent;
        }
    }
    else
    {
        print("trade failed! Card is not favaorable...  trade preference: "+preferenceResult);
    }

}
```

(Figure 13.3 Continent_AI script)

- Update city's index after a trade
    - Responsible for recalling updates for changing city index level after a successful trade operation in the previous turn

```
public void updateContinentIndexAfterTrade()
{
    if (isTradeCompleteInThisTurn)
    {
        updateIndex(completedTradeIndexUpdate[0], completedTradeIndexUpdate[1], completedTradeIndexUpdate[2], completedTradeWithThisContinent);
        isTradeCompleteInThisTurn = false;
    }

}
```

(Figure 13.4 Continent_AI script)

Global_Crisis

Responsible for controlling the global crisis events and perform corresponding updates on global and local

- Crisis initialisation
    - Responsible for creating a crisis event when called, as well as displaying crisis information on crisis panel in global

```
public void emergencyEventInitialise()
{
    isDuringEmergencyEvent = true;
    string continentName = "";
    int randomContinent = Random.Range(0, 5);
    switch (randomContinent){
        case 0:
            continentName = "South America";
            break;
        case 1:
            continentName = "North America";
            break;
        case 2:
            continentName = "Oceania";
            break;
        case 3:
            continentName = "Asia";
            break;
        case 4:
            continentName = "Europe";
            break;
        case 5:
            continentName = "Africa";
            break;
        default:
            break;
    }

    panel.SetActive(true);
    currentCrisis = crisisList[Random.Range(0, crisisList.Length)] ;
    Crisis_image.GetComponent<Image>().sprite = currentCrisis.Crisis_image;
    Emergency_text.text = "Continent: "+ continentName + "\n\n" +
        "Emergency type: " + currentCrisis.Emergency_type + "\n\n" +
        "Description: " + currentCrisis.Emergency_description + "\n\n" +
        "Require for: " + currentCrisis.Emergency_solve_method + "\n\n";
    Consequence_icon.GetComponent<Image>().sprite = currentCrisis.Consequence_icon;
    Consequence_text.text = "decrease by " + currentCrisis.Consequence_index + " globally if not resolved in 5 turns!";
    SDGicons.GetComponent<Image>().sprite = currentCrisis.E_SDG_Icons;
}
```

(Figure 14.1 Global_Crisis script)

- Crisis control
    - Responsible for monitoring and updating crisis conditions, such as determining how many turns the player have left for solving crisis, and updating global index when there are no turns left

```
public void emergencyUpdate()
{

    if (isDuringEmergencyEvent)
    {
        countdown--;

        if (countdown > 0)
        {
            //print("countdown is: " + countdown);
        }
        else
        {
            switch (currentCrisis.affectType)
            {
                case "econ":
                    globalScript.updateGlobalIndex(-currentCrisis.Consequence_index, 0, 0);
                    break;
                case "env":
                    globalScript.updateGlobalIndex(0, -currentCrisis.Consequence_index, 0);
                    break;
                case "soc":
                    globalScript.updateGlobalIndex(0, 0, -currentCrisis.Consequence_index);
                    break;
                default:
                    break;
            }
            countdown = 5;
            isDuringEmergencyEvent = false;
        }
    }
}
```

(Figure 14.2 Global_Crisis script)

- Crisis control

- Responsible for monitoring and updating crisis conditions, such as determining how many turns the player have left for solving crisis, and updating global index when there are no turns left, it will also call game_manager when needed for crisis message display.

```
public void emergencyUpdate()
{

    if (isDuringEmergencyEvent)
    {
        countdown--;

        if (countdown <= 0)
        {
            switch (currentCrisis.affectType)
            {
                case "econ":
                    globalScript.updateGlobalIndex(-currentCrisis.Consequence_index, 0, 0);
                    break;
                case "env":
                    globalScript.updateGlobalIndex(0, -currentCrisis.Consequence_index, 0);
                    break;
                case "soc":
                    globalScript.updateGlobalIndex(0, 0, -currentCrisis.Consequence_index);
                    break;
                default:
                    break;
            }
            countdown = 5;
            isDuringEmergencyEvent = false;
            requriedTurnsForCrisis = 3;

            isCrisisResolveStarted = false;

            mainScript.displayCrisisNews(3);
        }
    }
}
```

(Figure 14.2 Global_Crisis script)

- Crisis panel actions
    - Responsible for updating crisis controls and solving crisis functions based on Player's decision for agreeing or refusing to assist in global crisis, it will also call game_manager when needed for crisis message display.

```
public void onClickAccept()
{
    bool playerHasResolve = false;
    //check if player has sdg card
    for (int currentTileNum = 0; currentTileNum < 10; currentTileNum++)
    {
        TileScript occupiedBuildingPreset = mainScript.getTilePreset(currentTileNum);
        if(occupiedBuildingPreset != null)
        {
            if(currentCrisis.E_SDG_Icons.name == occupiedBuildingPreset.E_SDG_Icons.name)
            {
                //if already have one
                mainScript.displayCrisisNews(0);
                isCrisisResolveStarted = true;
                playerHasResolve = true;
            }
        }
    }
    if (!playerHasResolve)
    {
        //card creation if tile didnt have one already
        deckManager.crisisRefill(5);
        mainScript.displayCrisisNews(1);
    }

    panel.SetActive(false);
    mainScript.EmergencyEventEnder();
}

public void onClickReject()
{
    panel.SetActive(false);
    mainScript.EmergencyEventEnder();
}
```

(Figure 14.3 Global_Crisis script)

- Solving crisis
    - Responsible for updating crisis resolve conditions and monitor Player's actions in resolving crisis after a turn, it will also call game_manager when needed for crisis message display.

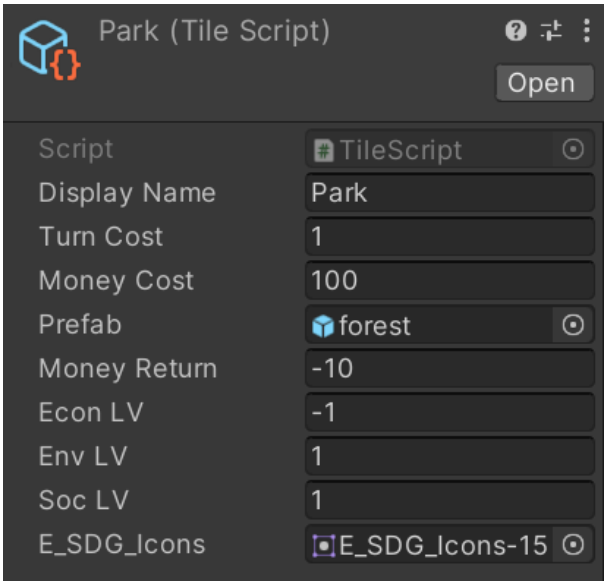
```
public void solvingCrisis()
{
    //check progress
    if (isDuringEmergencyEvent)
    {
        if (isCrisisResolveStarted)
        {

            requriedTurnsForCrisis--;
            if (requriedTurnsForCrisis <= 0)
            {
                //crisis adverted
                mainScript.displayCrisisNews(2);
                requriedTurnsForCrisis = 3;
                countdown = 5;
                isDuringEmergencyEvent = false;
                isCrisisResolveStarted = false;
            }
        }
        else
        {
            //check if have resolve tile in this turn
            for (int currentTileNum = 0; currentTileNum < 10; currentTileNum++)
            {
                TileScript occupiedBuildingPreset = mainScript.getTilePreset(currentTileNum);
                if (occupiedBuildingPreset != null)
                {
                    if (currentCrisis.E_SDG_Icons.name == occupiedBuildingPreset.E_SDG_Icons.name)
                    {
                        //if already have one
                        isCrisisResolveStarted = true;
                        requriedTurnsForCrisis--;
                        mainScript.displayCrisisNews(0);
                    }
                }
            }
        }
    }

}
```

(Figure 14.4 Global_Crisis script)

TileScript

Script for storing presets for buildings

- Preset information
    - Building name
    - Building turn cost (how many turns to complete)
    - Building money cost (how much money to required to be built)
    - Building prefab (game model)
    - Building maintenance cost / income for each turn
    - City index level for economic, environmental and social index (changes when built)
    - SDG icon image (correspond to the buildings’ usage)

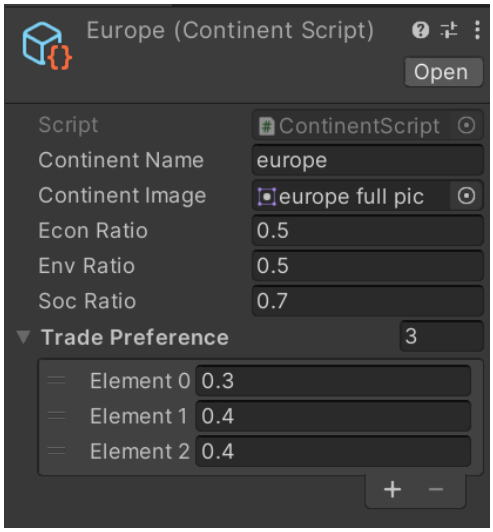


(Figure 15. Building preset informations)

ContinentScript

Script for storing presets for cities (continents)

- Preset information
  - Continent name
  - Continent image
  - Starting economic index
  - Starting environmental index
  - Starting social index
  - Trading Preference for economic, environmental and social index



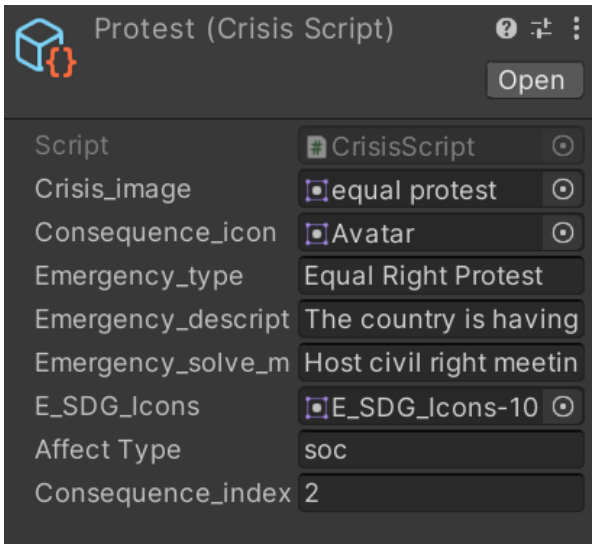
(Figure 16. Cities preset informations)

CrisisScript

Script for storing presets for crisis events

- Preset information
  - Crisis image
  - Crisis name

- Event description
- Resolve method description
- SDG icon image (correspond to the crisis)
- Affected index icon
- Affected index type
- Affected index value



(Figure 17. Crisis preset information)

2. In-Game Functions

Tile construction visualization

All player's possible interactions with tiles are visualised as following:

(Figure 18. Tiles)

- Red boxes represent a tile that is selected, but the player hasn't selected which building it needs to be built on. It will appear when the player's mouse interacts with the owned empty tile, and disappear and be replaced by yellow boxes once the player decides its building placement.
- Yellow box with the text "Construction in progress…" represent a tile that is planned for buildings, it will appear when the selected tile is under construction and will be replaced by building models once it's completed.
- Occupied tiles will be represented by building models, meaning the tile has a completed project building, player can activate building demolition function by clicking the occupied tile
- White tiles represent tiles that are currently not owned by player and can be purchased as owned empty tiles if player have enough money to buy it.
- Red tiles represented tiles that are currently not owned by player but is being selected for ownership purchase.

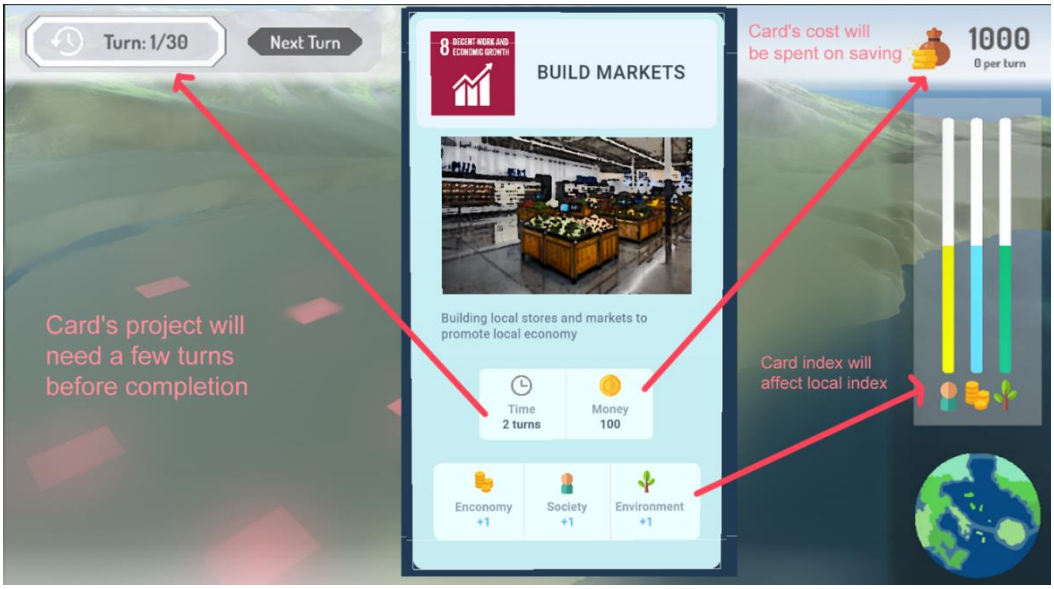
Next turn

With each turn progression, the total turns will increase by 1, while the player's saving will also be changed according to their maintenance cost / income from the buildings. Global panel will also be updated after a turn, such as updating the global index after a trade or a crisis, or starting a new crisis event

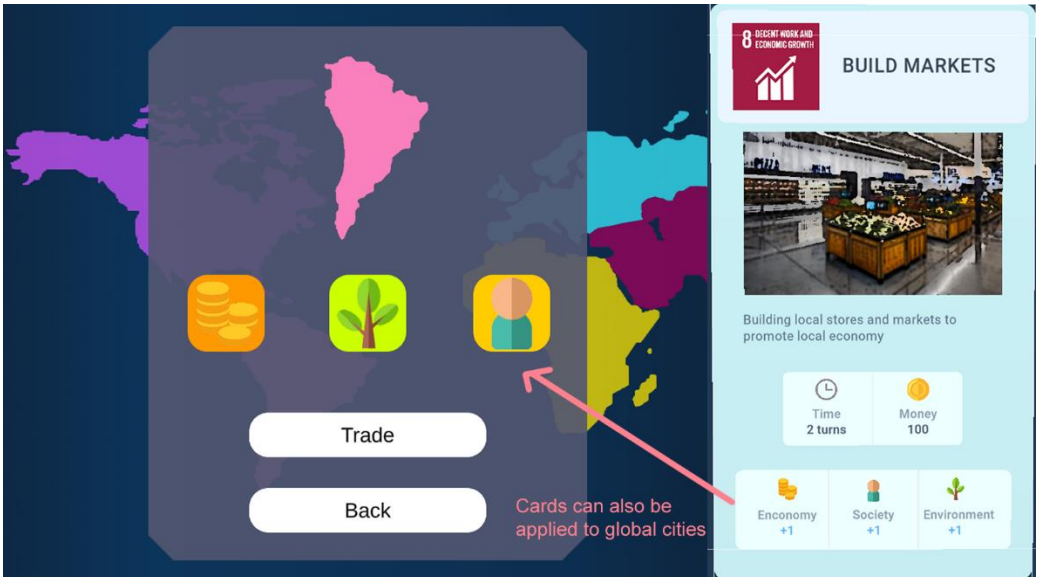
(Figure 19. Next turn)

Cards

Placing project cards would require certain turns and money (from player's bank saving). And would also affect local index based on the nature of the cards. Cards are also applicable to global city's indices after trading.



(Figure 20. A Sample Card)



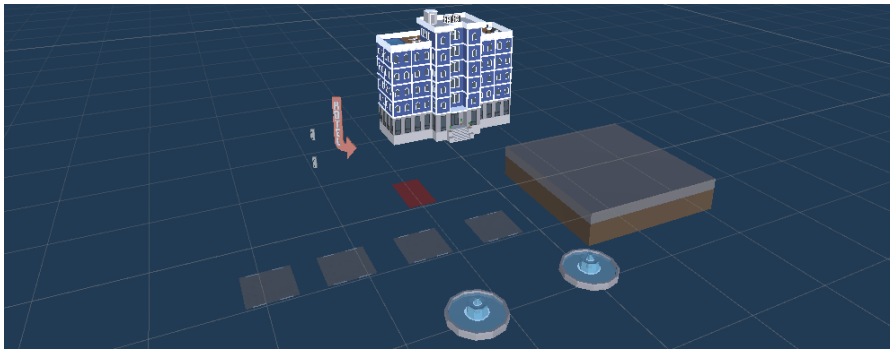
(Figure 21. Cards in global)

3. Game Models

Map tile

Each of the map tiles represent different categories of infrastructure in the real world.

- All tiles are created by combining sub-elements of different models:

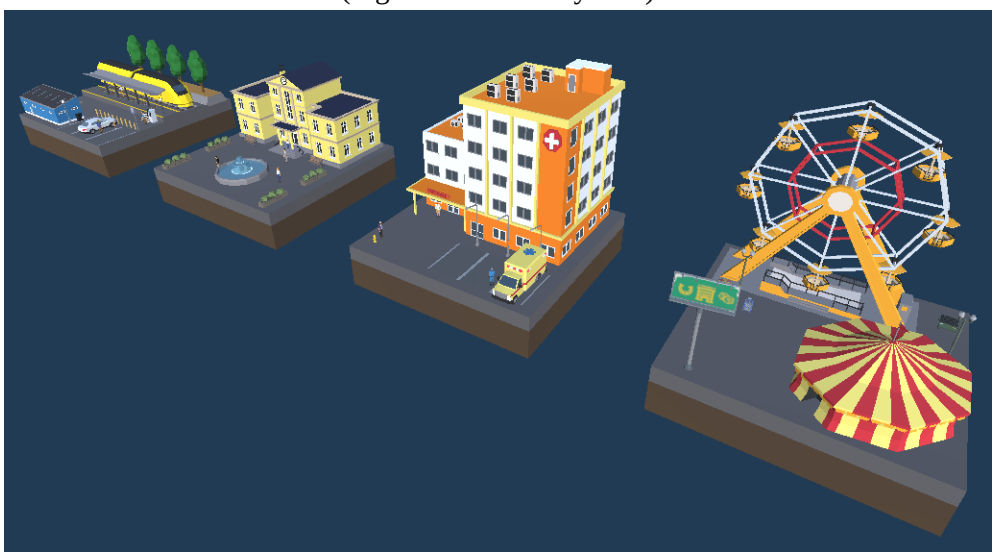


(Figure 22. Building dissections)

- Society tiles (represented in yellow colours)

Left to right: Transportation, Education, Medication, Entertainment

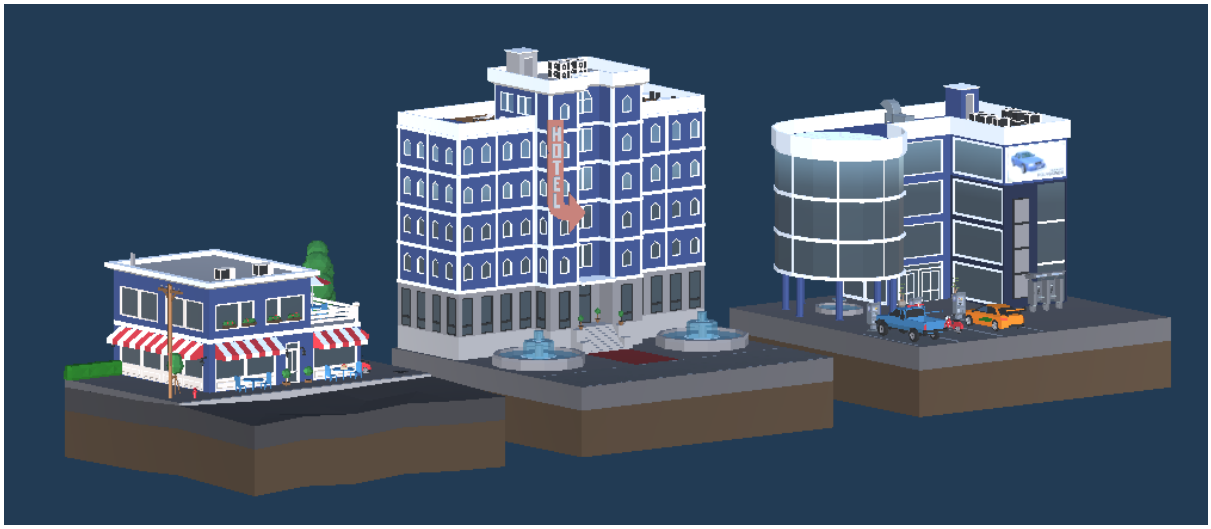
(Figure 23.1. Society tiles)



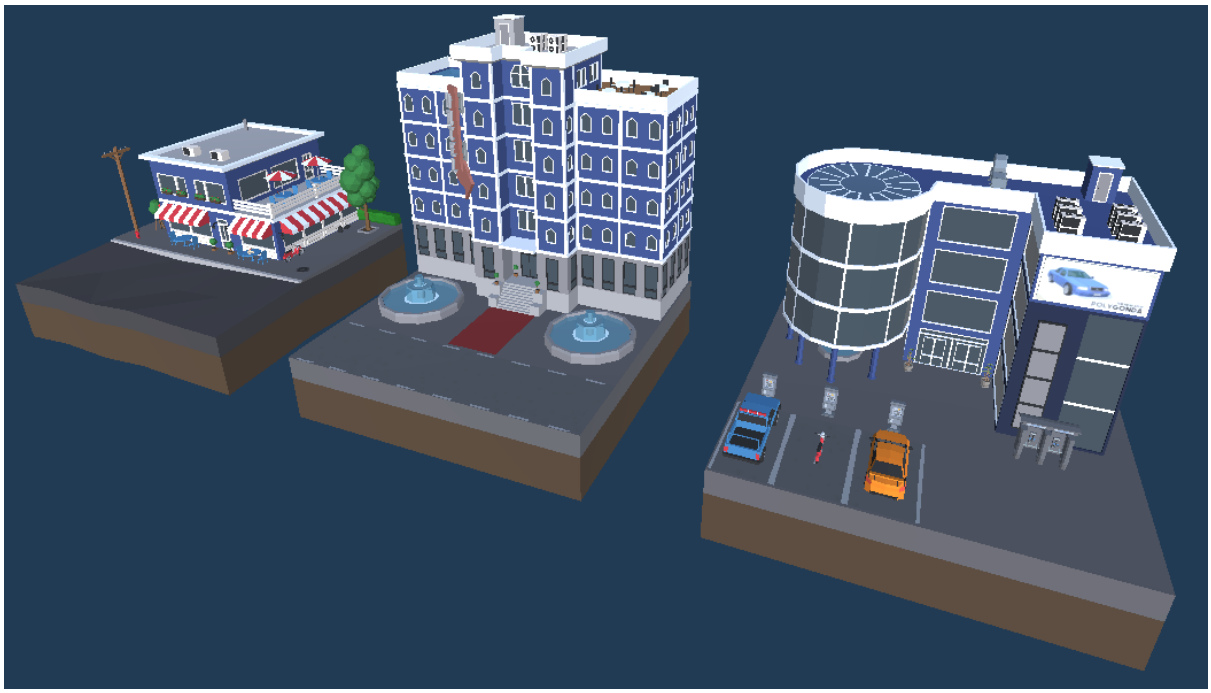
(Figure 23.2. Society tiles)

- Economic tiles (represented in blue colours)

Left to right: Shop, Hotel, Mall



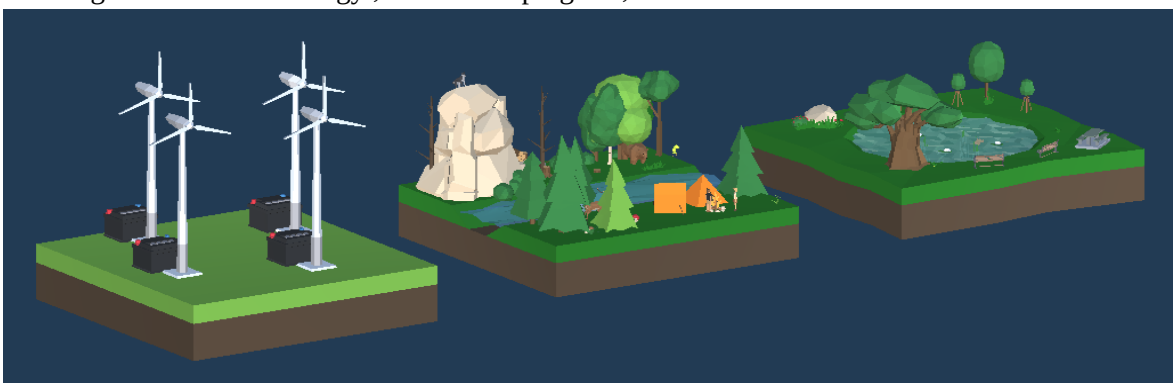
(Figure 24.1 Economic tiles)



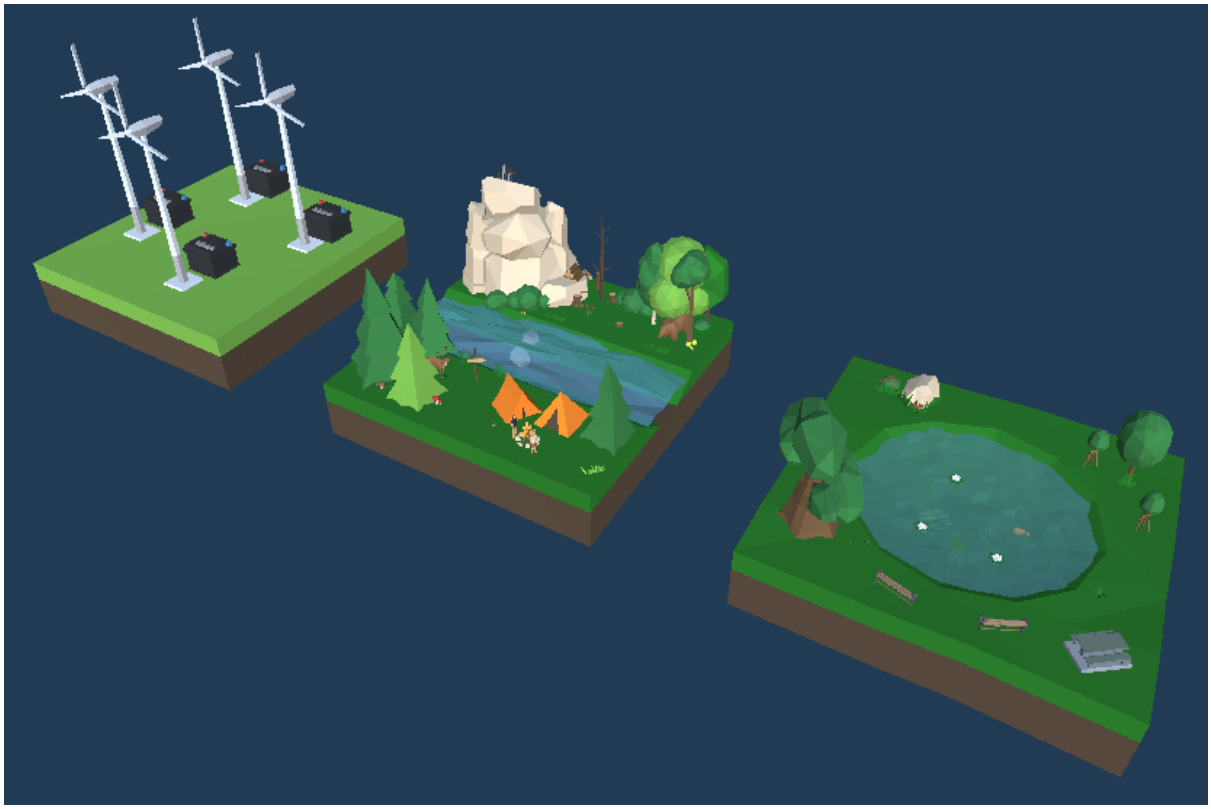
(Figure 24.2. Economic tiles)

- Environmental tiles (represented in green colours)

Left to right: Renewable Energy , Green Camping site, Environmental Conservation site



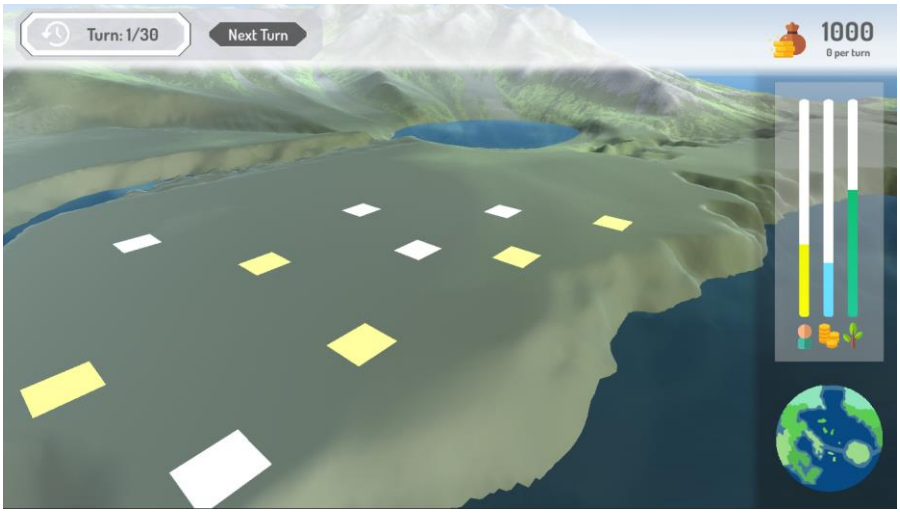
(Figure 25.1. Environmental tiles)



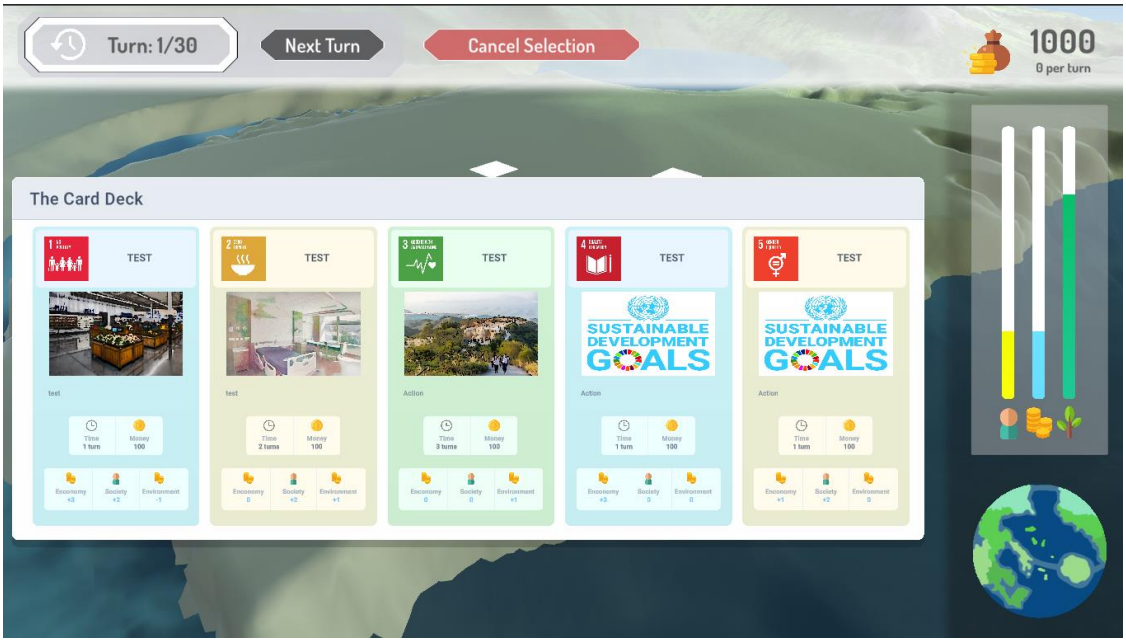
(Figure 25.1. Environmental tiles)

City Map

The main playground for players to plan their city, players can also observe the city's development index with the local index indicator on the right.
Players can select tiles for purchasing tiles or building infrastructures, project cards will pop-up once Player clicked owned tiles that are not occupied, they can also cancel tile selection from the button on top.
Players can initiate projects on their land using the action cards from the card deck on the bottom.
To enter the global panel, players will have to click the earth icon on the bottom right of the game ui.



(Figure 26.1. Game Map)



(Figure 26.2 Game Map when owned tiles are selected)

Global map

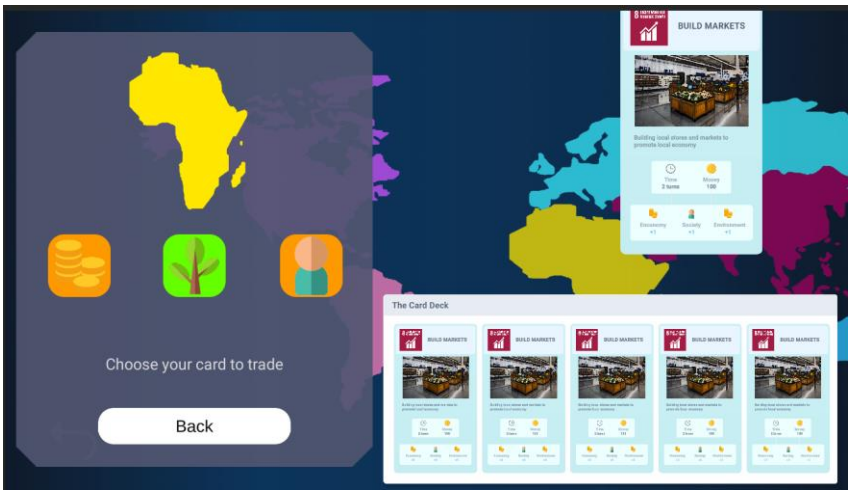
The global panel of the game, players can view other cities index level and engage trade with them. Clicking the continent image will open the city's information panel, which contains all 3 aspects of its index level. Upon clicking the trade button, trade panel will be shown. Player can exit the information panel or trade panel through the Quit / Back button in any time.

Players can choose which projects to trade from the card deck on the bottom after clicking trade button.

To return to the city map, players can click the arrow on the bottom left of the game ui.

(Figure 27.1. Global Map)

(Figure 27.2. Global Map)

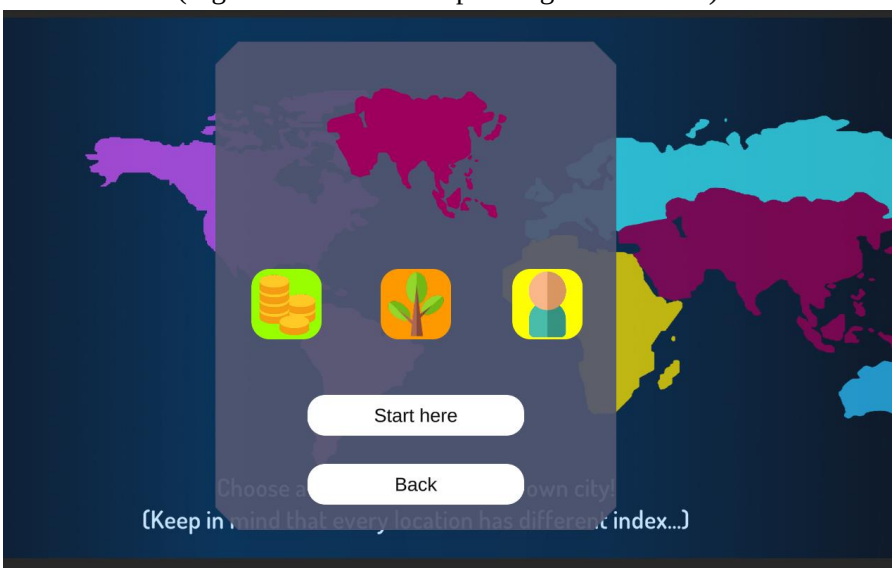


(Figure 27.3 Global Map)

Global map (during initialisation)

The global panel of the game will automatically appear at the start of the game, player will have to choose where their city should be located, clicking the continent will open the city's information panel, which contains all 3 aspects of its index level. Upon clicking the Start Here button, player will return to the city map with the corresponding continent's starting index levels. Player can exit the information panel through the Quit button in any time.

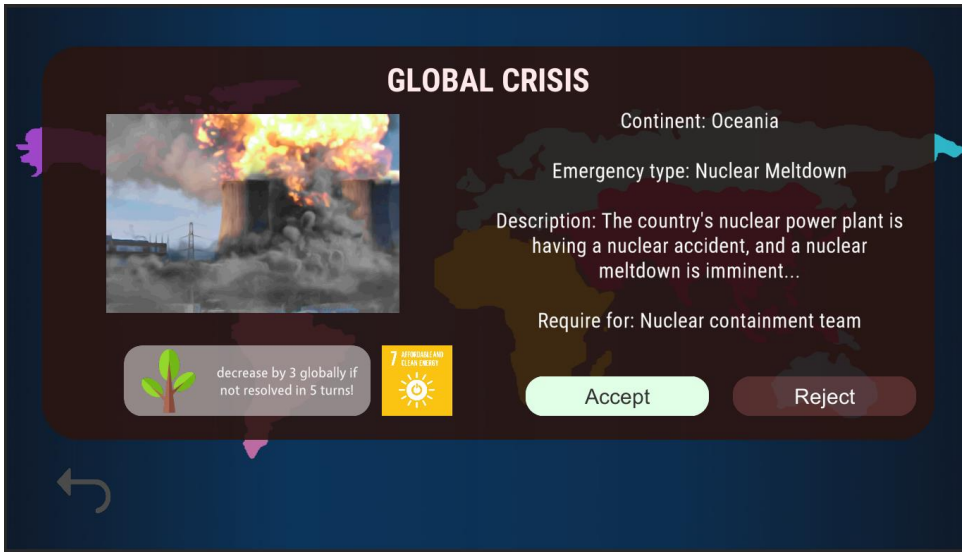
(Figure 28.1 Global Map during initialisation)



(Figure 28.2 Global Map during initialisation)

Global map (during crisis)

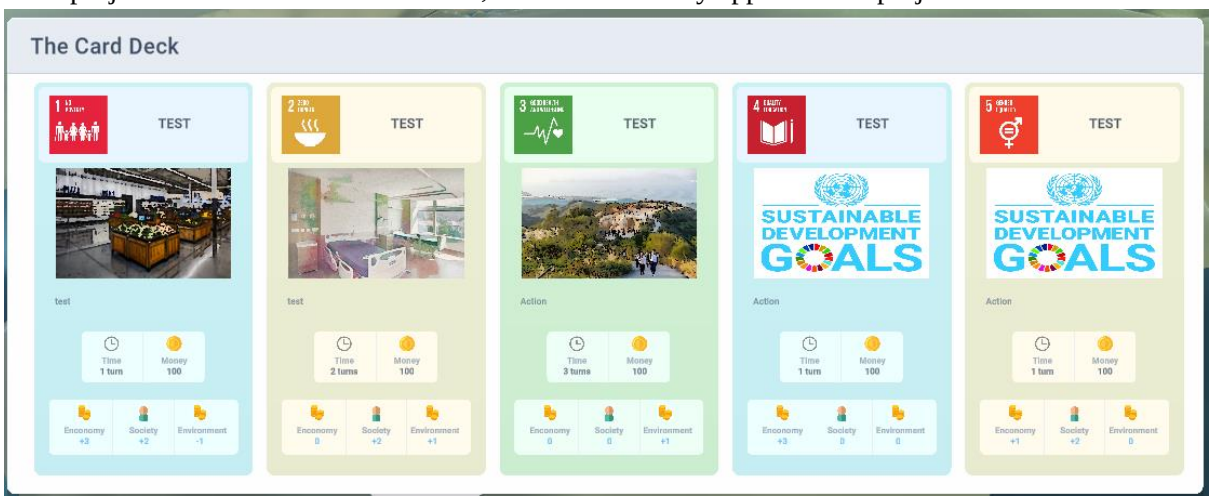
A crisis panel will automatically appear in global map in random occasions, and player should choose whether they accept or reject the crisis resolve proposal to avoid global negative impacts.



(Figure 29. Global Map during crisis)

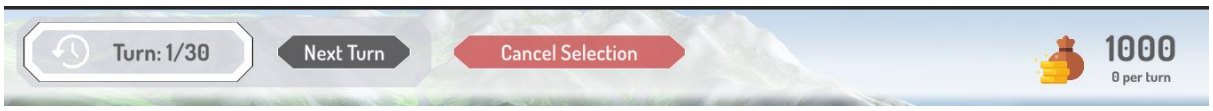
Action card

Each project have their own action cards, and will randomly appear in the project card deck

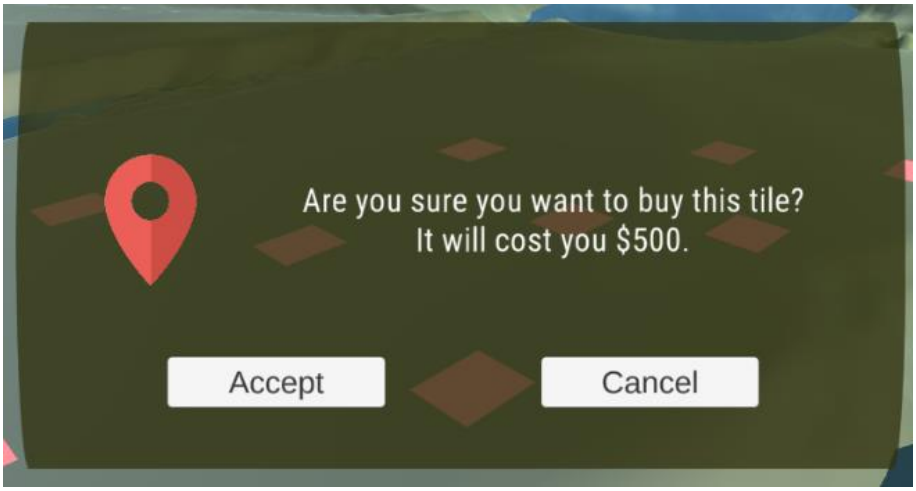


(Figure 30. Project Card Deck)

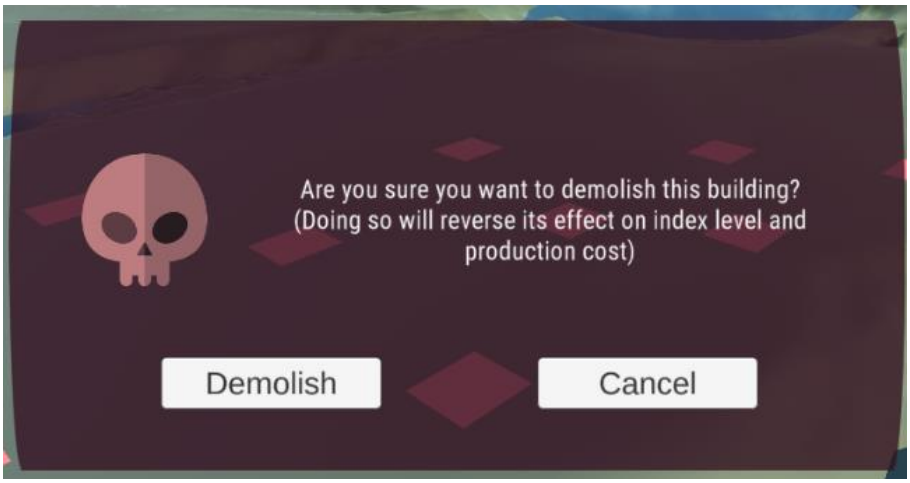
User interfaces



(Figure 31. Left to Right: Number of turns, Next turn function, cancel tile selection, Bank savings)



(Figure 32. Tile purchase panel)



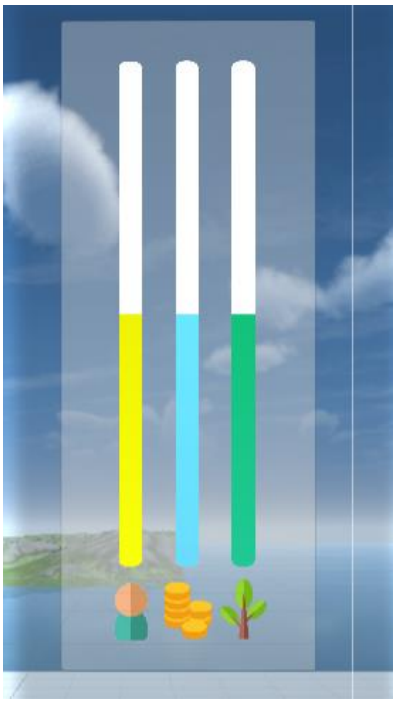
(Figure 33. Tile purchase panel)

(Figure 34.1. Crisis Update panel)

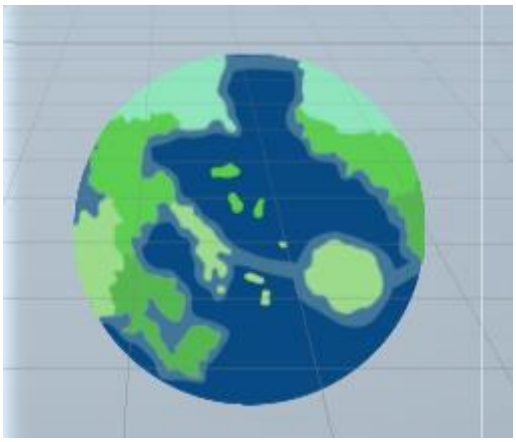
(Figure 34.2. Crisis Update panel)

(Figure 34.3. Crisis Update panel)

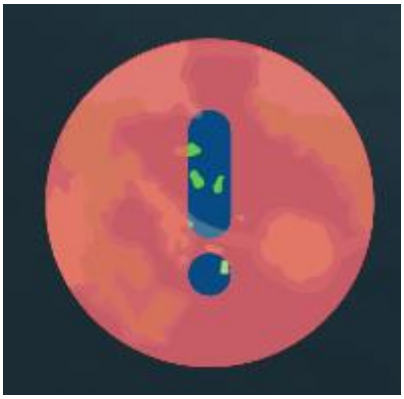
(Figure 34.4. Crisis Update panel)



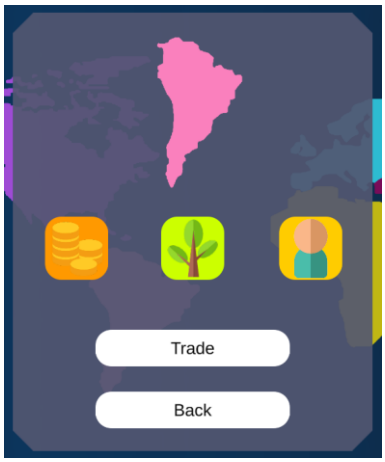
(Figure 35. Local index, Left to Right: Society, Economy, Environment)



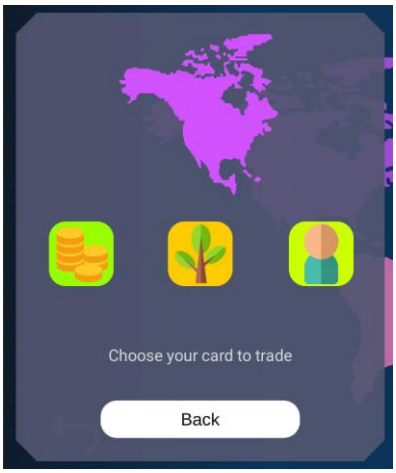
(Figure 36. Global Panel Icon)



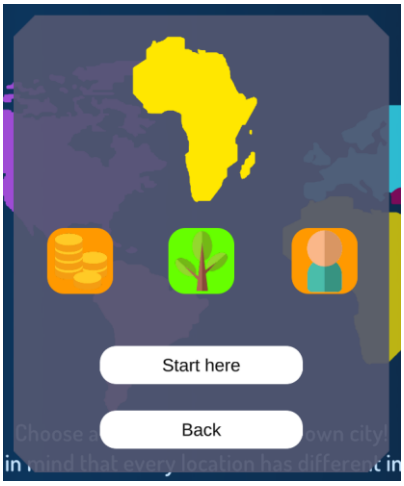
(Figure 37. Global Panel Icon during crisis)



(Figure 38. Global city information panel)

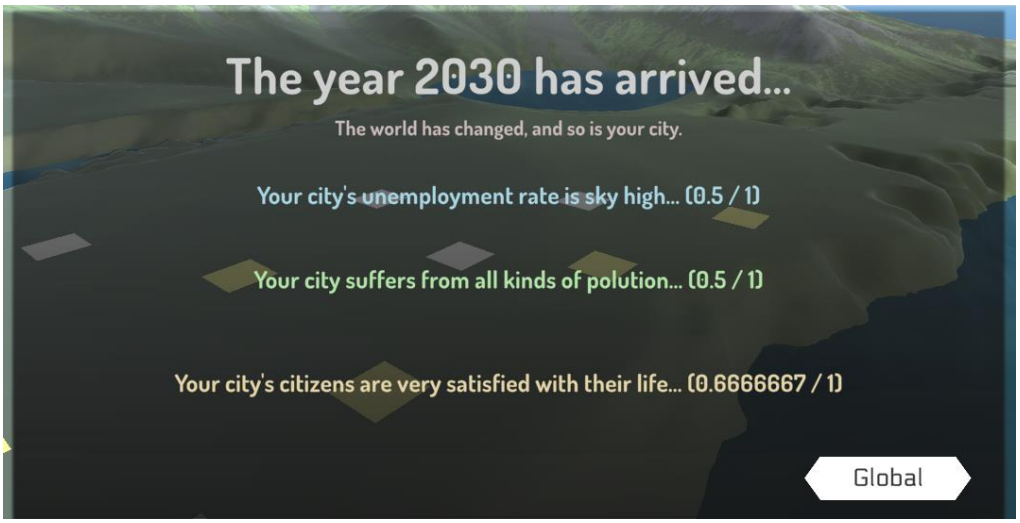


(Figure 39. Global city information panel during trading)



(Figure 40. Global city information panel during initialisation)

(Figure 41. Global crisis panel)



(Figure 42.1. Local Ending Panel)

(Figure 42.2. Global Ending Panel)

(Figure 42.3. Score Ending Panel)

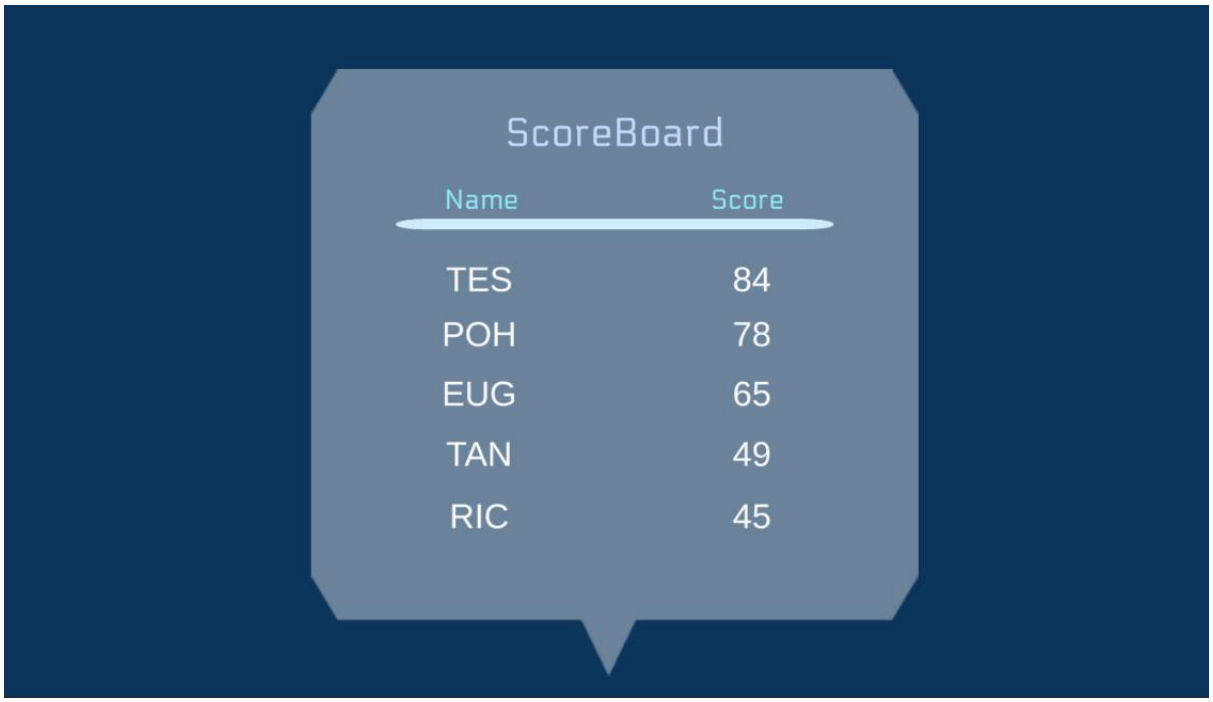
Menu

The main menu of the game.

(Figure 43. Main Menu)

Scoreboard

The scoreboard of the game, can be accessed from Menu, shows top ranking player's scores.



(Figure 44. Scoreboard)

References

General

[1] SDG 2030
https://2030sdgsgame.com/

[2] 孩子們的福爾摩沙 (Crisis management game)
https://npost.tw/archives/54429

In-game assets

World map from Global map
https://www.pikpng.com/pngvi/Jxbomb_asia-transparent-background-world-map-3d-clipart/

SDG icons
https://sdg.humanrights.dk/en/goals-and-targets

Game map terrain
https://assetstore.unity.com/packages/3d/environments/landscapes/realistic-terrain-collection-lite-47726#description

UI elements
https://assetstore.unity.com/packages/2d/gui/ultimate-clean-gui-pack-154574#description

In-game buildings and tiles
https://assetstore.unity.com/packages/3d/props/low-poly-ultimate-pack-54733#description