CT3536 Games Programming - Game Dev Project

Daniel Hannon (19484286)

January 2022

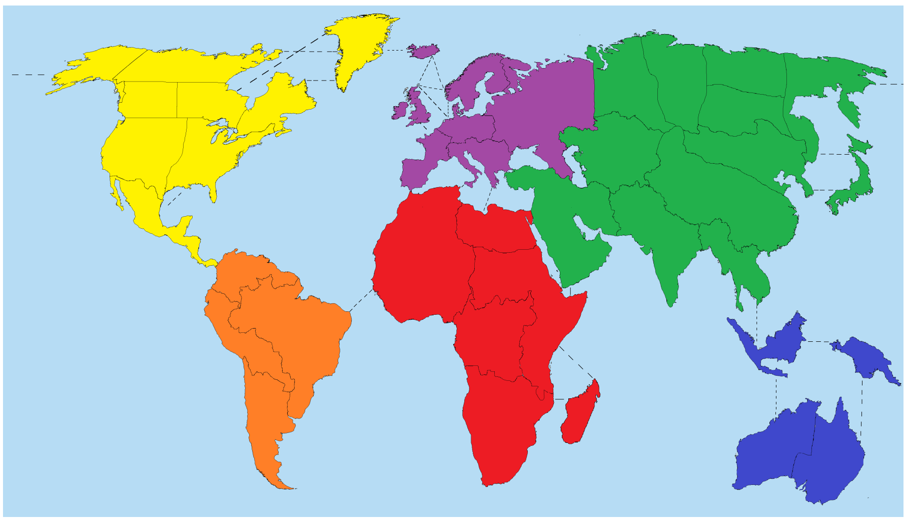
1 Overview

The game I wanted to make was Risk. The reason I wanted to do this was that I do not like the Steam Version of the game, linked here. A goal I made for myself was to make all of the assets (apart from the music, that was taken from Elder Scrolls IV Oblivion) here is a link to it. Overall I got most of what I wanted to done and this will be discussed more in the reflection.

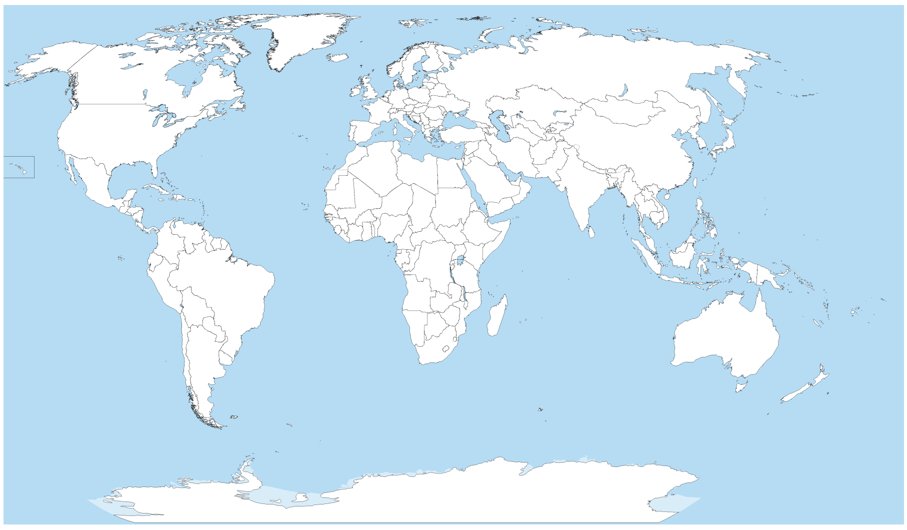
2 Creating the Assets

As I am not too experienced with creating assets, this was particularly challenging, I made the conscious decision to make the entire game 2D as I figured it would be easier to make 2D assets than 3D ones and it would contribute greatly to the aesthetic of the game.

2.1 Making The Map



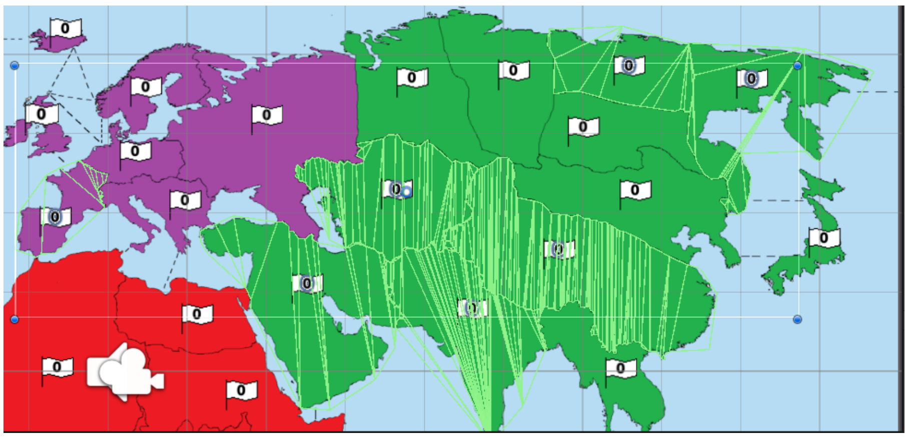
In Order to make a map that was as close to the risk board map as possible I needed something to work as a template, so I obtained a map from wikipedia (Source) that looked like this:
I had to perform numerous tasks to alter the map appropriately to have everything with the correct proportions



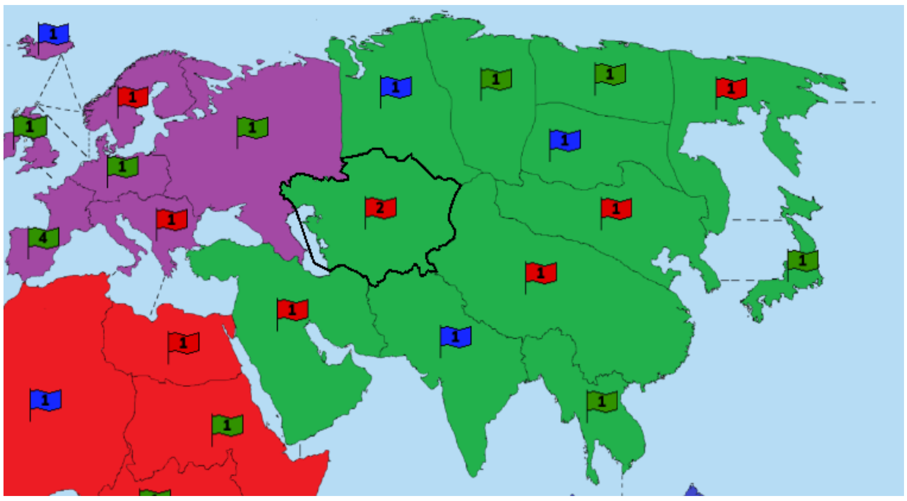
that included:

- Resizing Europe, Asia, & Africa
- Erasing Numerous Islands
- resizing greenland
- redrawing most borders to make the appropriate 42 regions,
- recoloring the territories to make the continents prominent

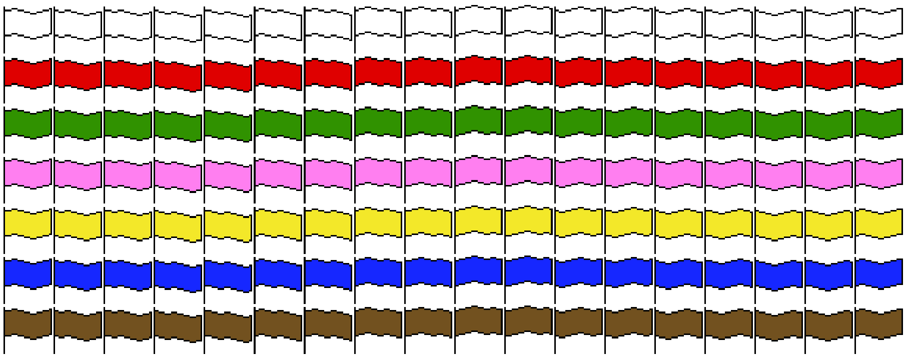
Once I had completed this, I needed to then make it so it was possible to click the region and for the game to know what it was. I achieved this by creating Polygon Colliders for every single Territory. This required me to manually make them follow the borders as closely as possible which was very time consuming. I utilized the special quality of islands and I was a bit more relaxed with how they are laid out and they are just irregular polygons (Unless they have a Land border where that is following the border as closely as possible) Here are some of them highlighted below: Then In order to get the Region The Mouse was over, I used RayCastHit2D to



see if I hit anything then performed some checks to see if the collision was valid.
I later realized that it was somewhat difficult to figure out what Territory I had clicked on So I made a Method for the Province Class that each province held that created a line renderer that followed the borders set out by the already existing polygon colliders as a nice visual reference for the player. Here is a demonstration where I click on afghanistan: This screenshot leads to the next section



2.2 Making the flags

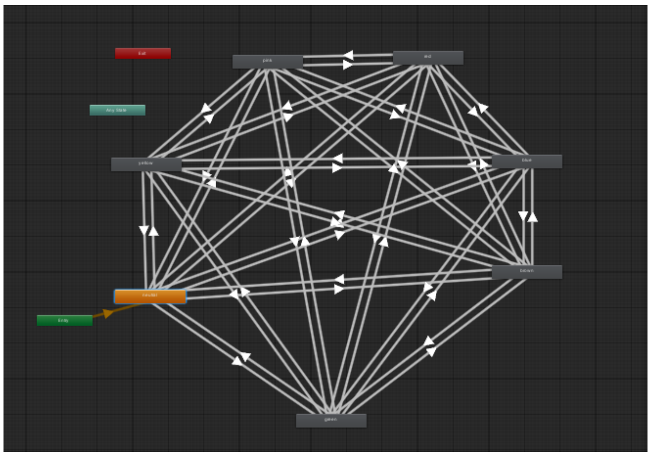


In order to make the flags I used GIMP to make a 32x32 flag, I wanted to make it animated so I created a Flag waving animation to the best of my ability. So Then I duplicated it seven times and added unique colors. I then made seven Unity animations using this spritesheet I created. I was now faced with a new Issue which leads me to the next point.

2.3 Changing the flags color

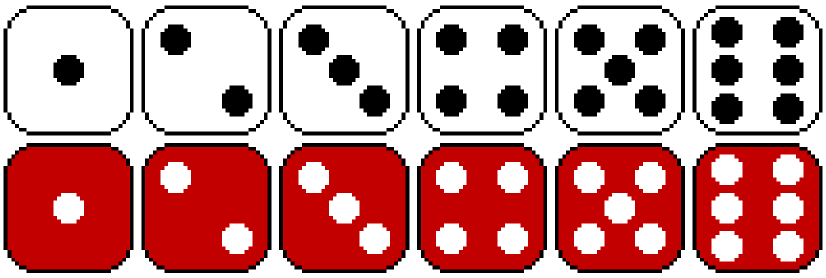
Abandon all hope, ye who enter here

In order to make it transition between the flags when changing teams I had to make an animation tree, sadly



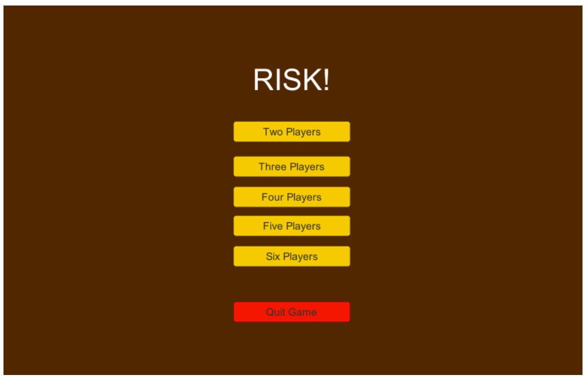
AnyState would not allow the transitions to match the behavior I wanted(They kept transitioning mid animation) I hard coded all 36 conversions. The way I did this was I made an AnimationController with one varible: Team, this was kept as an integer and each value corresponds to a different flag and then if the value changes, it goes to the appropriate flag and so on.

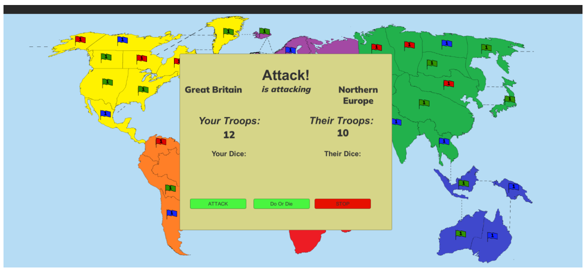
2.4 Making the Dice

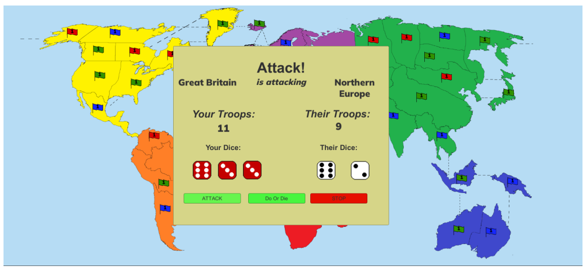


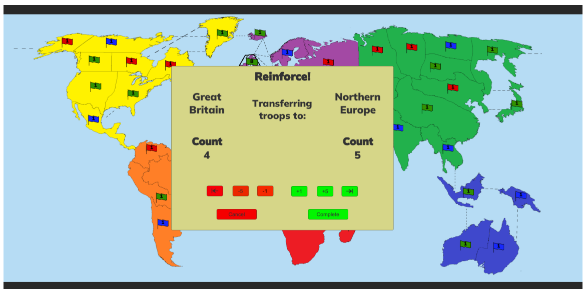
I made a 32x32 dice in GIMP and then made all six faces and then an alternative color for the attack dice

3 Making the menus









The Menus were very straight forward for the most part, as they were primarily consisted of buttons, apart from teh dice. The Dice were made by making panels with dice faces as images. Each Dice was attached to a Dice Class that was made and when you Invoked Dice.Roll() it automatically updated the faces of the dice.

4 The Game Itself

Now once all the UI has been shown. I will now go over the turn logic

4.1 Setup

As Risk has two ways of setting up (Either the players pick territories one by one, or they are distributed fairly at random), I Opted for the distribution at random option as I figured that it would be easier to implement. I do however, still have to allow them to populate these regions one by one at the end of the setup phase.

4.2 Your Turn

4.2.1 Replenish

In order to replenish a calculation must be performed to get the number of troops you can distribute. This is done in the same way as the physical variant of risk. That being that the amount of territories you possess are divide by three and rounded down and this value or three whichever is greater, is then added to contenential bonuses (These are done by having each continent in a list and I check to see if that list is a subset of all the territories owned by a given player, if it is then the award is granted). You then have to distribute the troops you were awarded here before the beginning of the next phase.

4.3 Attack

The attack form stage is fairly straightforward, you click on a territory if it belongs to you it becomes marked. Then if you selected a territory that is not your own afterwards, an adjacency check is performed and if it is not met then you cannot attack, otherwise the attack interface shows up. If you happen to click on a territory of your own after already selecting a territory of your own, the previously selected territory is replaced with this. If you win the battle a reinforcement interface shows up which allows you to move troops to your newly conquered territory without ending your turn.

4.4 Reinforce

this is the final phase of the Turn. This allows you to move units from one territory to another so long as some connected path exists, and you own both territories. A path is calculated using a simple Depth First Search that I made and then if a path exists then the reinforcement interface shows up, and when this reinforcement ends, your turn concludes. A check is performed to see if anyone owns all 42 territories and if they do you are booted to the menu and the game concludes.

4.5 What is not here/issues

I struggled to calculate an appropriate timeframe for the game and ultimately the UI Design took far longer than I anticipated, so I ended up making this a local hotseat game as I did not have enough time to implement AI. All else that Is missing is a territory card system.
Additionally, the game appears somewhat squashed on some monitors as I forgot to account for a 16:9 Aspect Ratio and I ended up forcing a 23:9 ratio to ensure that the entire map was visible.

5 Reflection

Overall this was a very educational experience where I learned a lot about games and project management. I am tempted to keep working on the game and try implement an AI as I reckon it would be pretty fun to do.

6 Code

6.1 Game Manager Script

```csharp
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;


/*
 *  GameManagerScript.cs
 *  Author: Daniel Hannon (19484286)
 *  Version: 1
 */
public class GameManagerScript : MonoBehaviour {
  // Game Control Variables
  public Camera mainCamera;
  public bool InDebugMode;
  private bool attackMenuVisible = false;
  private bool reinforceMenuVisible = false;
  private bool mainMenuVisible = true;
  public Text statusBar;
  public Button EndTurnButton;

  //Selectors
  public GameObject selected;
  public GameObject selected2;

  //Menus
  public GameObject atkmenu;
  public GameObject renMenu;
  public GameObject statusMenu;
  public GameObject mainMenu;

  //Main Menu Buttons
  public Text MenuText;
  public Button twoPlayerButton;
public Button threePlayerButton;
public Button fourPlayerButton;
public Button fivePlayerButton;
public Button sixPlayerButton;
public Button quitGame;

//Attack Menu Specific Variables
public Text attacker;
public Text defender;
public Text attackerTextCount;
public Text defenderTextCount;
public GameObject[] AttackerDie;
public GameObject[] DefenderDie;
public Button AttackButton;
public Button DoOrDieButton;
public Button CancelButton;

//Reinforce Menu Specific Variables
public Text ReinforceFrom;
public Text ReinforceTo;
public Text ReinforceFromCount;
public Text ReinforceToCount;
private int FromTerritoryTemp = 0;
private int ToTerritoryTemp = 0;
public Button reinforceCancel;
public Button reinforceConfirm;
public Button reinforceAdd1;
public Button reinforceDec1;
public Button reinforceAdd5;
public Button reinforceDec5;
public Button reinforceAddAB1;
public Button reinforceDecAB1;

//Turn Specific stuff
enum Stages { NOT_STARTED, GAME_SETUP, REPLENISH, ATTACK, REINFORCE };
Stages currStage = Stages.NOT_STARTED;
int currentPlayer = 0;
int numberOfActivePlayers = 2;
int numberOfTroopsToPlace = 0;
public List<string> playerColors;
List<Province>[] TeamTerritories = new List<Province>[6];
public bool reinforcementPerformedThisTurn = false;


// Essential Game Variables
// I initalize everything at runtime so I don't need to constantly allocate/deallocate memory
public List<Province> ClosedList = new List<Province>();
public List<Province> OpenList = new List<Province>();
public List<string> TeamColors = new List<string>() { "neutral", "red", "blue", "green",
↪  "yellow", "pink", "brown"};
public List<Province> AllProvinces = new List<Province>();
public List<Province> Europe = new List<Province>();
public List<Province> Africa = new List<Province>();
public List<Province> Asia = new List<Province>();
public List<Province> Oceania = new List<Province>();
public List<Province> SouthAmerica = new List<Province>();
public List<Province> NorthAmerica = new List<Province>();

void Start() {
  mainCamera.aspect = 23f / 9f;
```

```
    //Attack Menu Setup
    atkmenu.SetActive(false);
    AttackButton.GetComponent<Button>().onClick.AddListener(PerformAttack);
    CancelButton.GetComponent<Button>().onClick.AddListener(CancelButtonListener);
    DoOrDieButton.GetComponent<Button>().onClick.AddListener(DoOrDieListener);

    //Reinforce Menu Setup
    renMenu.SetActive(false);
    reinforceCancel.GetComponent<Button>().onClick.AddListener(CancelReinforce);
    reinforceAdd1.GetComponent<Button>().onClick.AddListener(ReinforceIncrement1);
    reinforceAdd5.GetComponent<Button>().onClick.AddListener(ReinforceIncrement5);
    reinforceAddAB1.GetComponent<Button>().onClick.AddListener(ReinforceIncrementAB1);
    reinforceDec1.GetComponent<Button>().onClick.AddListener(ReinforceDecrement1);
    reinforceDec5.GetComponent<Button>().onClick.AddListener(ReinforceDecrement5);
    reinforceDecAB1.GetComponent<Button>().onClick.AddListener(ReinforceDecrementAB1);
    reinforceConfirm.GetComponent<Button>().onClick.AddListener(ReinforceConfirm);

    EndTurnButton.GetComponent<Button>().onClick.AddListener(NewTurn);
    statusMenu.SetActive(false);

    twoPlayerButton.GetComponent<Button>().onClick.AddListener(TwoPlayer);
    threePlayerButton.GetComponent<Button>().onClick.AddListener(ThreePlayers);
    fourPlayerButton.GetComponent<Button>().onClick.AddListener(FourPlayers);
    fivePlayerButton.GetComponent<Button>().onClick.AddListener(FivePlayers);
    sixPlayerButton.GetComponent<Button>().onClick.AddListener(SixPlayers);
    quitGame.GetComponent<Button>().onClick.AddListener(QuitGame);

    InDebugMode = false;
  }

  // Attack Mechanism Listeners

  void AttackSetup() {
    if (CheckAdjacency(selected.GetComponent<Province>(),
    ↪   selected2.GetComponent<Province>()) &&
    ↪   !selected.GetComponent<Province>().Color.Equals(selected2.GetComponent<Province>().C
    ↪   {
      attackMenuVisible = !attackMenuVisible;
      atkmenu.SetActive(attackMenuVisible);
      attacker.text = selected.GetComponent<Province>().ProvinceName;
      defender.text = selected2.GetComponent<Province>().ProvinceName;
      attackerTextCount.text = selected.GetComponent<Province>().TroopCount.ToString();
      defenderTextCount.text = selected2.GetComponent<Province>().TroopCount.ToString();
    }
  }

  void CancelButtonListener() {
    for(int i = 0; i < 3; i++) {
      AttackerDie[i].GetComponent<DiceRoller>().SetInactive();
    }

    for(int j = 0; j < 2; j++) {
      DefenderDie[j].GetComponent<DiceRoller>().SetInactive();

    }
    attackMenuVisible = false;
    atkmenu.SetActive(false);
  }

  void DoOrDieListener() {
    //Calls Perform Attack until either you conquer the territory, or run out of troops, whatever come
    while(attackMenuVisible) {
      PerformAttack();
    }
  }

  void PerformAttack() {
    //Basic Attack Code lol
    int AttackerDieCount = 0;
    int DefenderDieCount = 0;
    List<int> AttackerDiceRolls = new List<int>();
    List<int> DefenderDiceRolls = new List<int>();
    //Get Number of Attacker Die
    switch(selected.GetComponent<Province>().TroopCount) {
      case 1:
        CancelButtonListener();
        break;
      case 2:
        //One Die
        AttackerDieCount = 1;
        break;
      case 3:
        //Two Die
        AttackerDieCount = 2;
        break;
      default:
        //Three Die
        AttackerDieCount = 3;
        break;
    }

    //Get Number of Defender Die
    switch(selected2.GetComponent<Province>().TroopCount) {
      case 0:
        string temp = selected2.GetComponent<Province>().Color;
        selected2.GetComponent<Province>().Color =
        ↪   selected.GetComponent<Province>().Color;
        selected2.GetComponent<Province>().TroopCount = 1;
        selected.GetComponent<Province>().TroopCount--;
        selected2.GetComponent<Province>().Flag.GetComponent<Animator>().SetInteger("team",
        ↪   TeamColors.IndexOf(selected2.GetComponent<Province>().Color));
        TeamTerritories[playerColors.IndexOf(selected.GetComponent<Province>().Color)].Add(s
        TeamTerritories[playerColors.IndexOf(temp)].Remove(selected2.GetComponent<Province>(
        CancelButtonListener();
        if(selected.GetComponent<Province>().TroopCount > 1) {
          ReinforceSetup();
        }
        break;
      case 1:
        DefenderDieCount = 1;
        break;
      default:
        DefenderDieCount = 2;
        break;
    }

    for(int i = AttackerDieCount; i < 3; i++) {
      AttackerDie[i].GetComponent<DiceRoller>().SetInactive();
    }

    for (int i = DefenderDieCount; i < 2; i++) {
      DefenderDie[i].GetComponent<DiceRoller>().SetInactive();
    }

    for(int i = 0; i < AttackerDieCount; i++) {
      AttackerDiceRolls.Add(AttackerDie[i].GetComponent<DiceRoller>().Roll());
    }

    for(int i = 0; i < DefenderDieCount; i++) {
      DefenderDiceRolls.Add(DefenderDie[i].GetComponent<DiceRoller>().Roll());
    }

    AttackerDiceRolls.Sort();
    AttackerDiceRolls.Reverse();
    DefenderDiceRolls.Sort();
    DefenderDiceRolls.Reverse();

    while(!(AttackerDiceRolls.Count == 0) && !(DefenderDiceRolls.Count == 0)) {
      if(DefenderDiceRolls[0] >= AttackerDiceRolls[0]) {
        selected.GetComponent<Province>().TroopCount--;
        attackerTextCount.text = selected.GetComponent<Province>().TroopCount.ToString();
      } else {
        selected2.GetComponent<Province>().TroopCount--;
        defenderTextCount.text = selected2.GetComponent<Province>().TroopCount.ToString();
      }
      AttackerDiceRolls.RemoveAt(0);
      DefenderDiceRolls.RemoveAt(0);
    }
  }

  // Troop Transfer/ Reinforce Mechanism

  public void ReinforceSetup() {
    if (CheckForPath(selected.GetComponent<Province>(), selected2.GetComponent<Province>())
    ↪  &&
    ↪  selected2.GetComponent<Province>().Color.Equals(selected.GetComponent<Province>().Co
    ↪  {
      reinforcementPerformedThisTurn = true;
      reinforceMenuVisible = true;
      renMenu.SetActive(true);
      ReinforceFrom.text = selected.GetComponent<Province>().ProvinceName;
      ReinforceTo.text = selected2.GetComponent<Province>().ProvinceName;
      FromTerritoryTemp = selected.GetComponent<Province>().TroopCount - 1;
      ToTerritoryTemp = 0;
      ReinforceFromCount.text = (selected.GetComponent<Province>().TroopCount -
      ↪  ToTerritoryTemp).ToString();
      ReinforceToCount.text = selected2.GetComponent<Province>().TroopCount.ToString();
    }
  }

  public void ReinforceIncrement1() {
    if (FromTerritoryTemp >= 1) {
      FromTerritoryTemp -= 1;
      ToTerritoryTemp++;
      ReinforceFromCount.text = (selected.GetComponent<Province>().TroopCount -
      ↪  ToTerritoryTemp).ToString();
      ReinforceToCount.text = (selected2.GetComponent<Province>().TroopCount +
      ↪  ToTerritoryTemp).ToString();
    }
  }
```

```
public void ReinforceDecrement1() {
  if (ToTerritoryTemp >= 1) {
    ToTerritoryTemp -= 1;
    FromTerritoryTemp++;
    ReinforceFromCount.text = (selected.GetComponent<Province>().TroopCount -
    ↪  ToTerritoryTemp).ToString();
    ReinforceToCount.text = (selected2.GetComponent<Province>().TroopCount +
    ↪  ToTerritoryTemp).ToString();
  }
}

public void ReinforceIncrement5() {
  if (FromTerritoryTemp >= 1) {
    if (FromTerritoryTemp >= 5) {
      FromTerritoryTemp -= 5;
      ToTerritoryTemp += 5;

    } else {
      ToTerritoryTemp += FromTerritoryTemp;
      FromTerritoryTemp = 0;
    }
    ReinforceFromCount.text = (selected.GetComponent<Province>().TroopCount -
    ↪  ToTerritoryTemp).ToString();
    ReinforceToCount.text = (selected2.GetComponent<Province>().TroopCount +
    ↪  ToTerritoryTemp).ToString();
  }
}

public void ReinforceDecrement5() {
  if (ToTerritoryTemp >= 1) {
    if (ToTerritoryTemp >= 5) {
      ToTerritoryTemp -= 5;
      FromTerritoryTemp += 5;

    }
    else {
      FromTerritoryTemp += ToTerritoryTemp;
      ToTerritoryTemp = 0;
    }
    ReinforceFromCount.text = (selected.GetComponent<Province>().TroopCount -
    ↪  ToTerritoryTemp).ToString();
    ReinforceToCount.text = (selected2.GetComponent<Province>().TroopCount +
    ↪  ToTerritoryTemp).ToString();
  }
}

public void ReinforceIncrementAB1() {
  ToTerritoryTemp += FromTerritoryTemp;
  FromTerritoryTemp = 0;
  ReinforceFromCount.text = (selected.GetComponent<Province>().TroopCount -
  ↪  ToTerritoryTemp).ToString();
  ReinforceToCount.text = (selected2.GetComponent<Province>().TroopCount +
  ↪  ToTerritoryTemp).ToString();
}

public void ReinforceDecrementAB1() {
  FromTerritoryTemp += ToTerritoryTemp;
  ToTerritoryTemp = 0;
  ReinforceFromCount.text = (selected.GetComponent<Province>().TroopCount -
  ↪  ToTerritoryTemp).ToString();
```

```
    ReinforceToCount.text = (selected2.GetComponent<Province>().TroopCount +
    ↪   ToTerritoryTemp).ToString();
  }

  public void CancelReinforce() {
    //Close without making change
    reinforceMenuVisible = false;
    renMenu.SetActive(false);
  }

  public void ReinforceConfirm() {
    selected.GetComponent<Province>().TroopCount -= ToTerritoryTemp;
    selected2.GetComponent<Province>().TroopCount += ToTerritoryTemp;

    FromTerritoryTemp = 0;
    ToTerritoryTemp = 0;
    reinforceMenuVisible = false;
    renMenu.SetActive(false);
  }

  //Main Menu
  public void TwoPlayer() {
    numberOfActivePlayers = 2;
    mainMenu.SetActive(false);
    mainMenuVisible = false;
    statusMenu.SetActive(true);
    GameSetup();
  }

  public void ThreePlayers() {
    numberOfActivePlayers = 3;
    mainMenu.SetActive(false);
    mainMenuVisible = false;
    statusMenu.SetActive(true);
    GameSetup();
  }

  public void FourPlayers() {
    numberOfActivePlayers = 4;
    mainMenu.SetActive(false);
    mainMenuVisible = false;
    statusMenu.SetActive(true);
    GameSetup();
  }

  public void FivePlayers() {
    numberOfActivePlayers = 5;
    mainMenu.SetActive(false);
    mainMenuVisible = false;
    statusMenu.SetActive(true);
    GameSetup();
  }

  public void SixPlayers() {
    numberOfActivePlayers = 6;
    mainMenu.SetActive(false);
    mainMenuVisible = false;
    statusMenu.SetActive(true);
    GameSetup();
  }
```

```

public void QuitGame() {
  Application.Quit();
}

//General Game Mechanics

public GameObject GetTerritoryClick() {
  Vector2 worldPoint = Camera.main.ScreenToWorldPoint(Input.mousePosition);
  RaycastHit2D hit = Physics2D.Raycast(worldPoint, Vector2.zero);
  if(hit) {
    if (hit.collider.gameObject.layer == LayerMask.NameToLayer("Provinces")) {
      return hit.collider.gameObject;
    }
  }
  return null;
}

public bool CheckAdjacency(Province p1, Province p2) {
  //this is for attacking
  for(int i = 0; i < p1.neighbors.Length; i++) {
    if (p1.neighbors[i].GetComponent<Province>().Equals(p2)) {
      return true;
    }
  }
  return false;
}

public bool CheckForPath(Province p1, Province p2) {
  OpenList.Add(p1);
  return CheckForPath(p2);
}

public bool CheckForPath(Province p2) {
  //this is for reinforcement
  Province temp;
  Province temp2;
  while(OpenList.Count != 0) {
    temp = OpenList[0];
    OpenList.Remove(temp);
    ClosedList.Add(temp);
    for (int i = 0; i < temp.neighbors.Length; i++) {
      temp2 = temp.neighbors[i].GetComponent<Province>();
      if(temp2.Color.Equals(temp.Color)) {
        if (temp2.Equals(p2)) {
          OpenList.Clear();
          ClosedList.Clear();
          return true;
        } else if (ClosedList.Contains(temp2) != true) {
          OpenList.Add(temp2);
        }
      }
    }
  }
  OpenList.Clear();
  ClosedList.Clear();
  return false;
}

public void DebugEval() {
  if (Input.GetKeyDown(KeyCode.E)) {
    GameSetup();
  }


  if (Input.GetMouseButtonDown(0) && !attackMenuVisible && !reinforceMenuVisible) {
    GameObject temp = GetTerritoryClick();
    if (temp != null) {
      if (selected != null) {
        if (temp != selected) {
          selected.GetComponent<Province>().Deselect();
        }
      }
      if (temp != selected) {
        temp.GetComponent<Province>().Select();
      }
      selected = temp;
      selected.GetComponent<Province>().TroopCount += 1;
    }
  }

  if (Input.GetMouseButtonDown(1) && !attackMenuVisible) {
    selected2 = GetTerritoryClick();
  }

  if (selected != null) {
    /*
     * Neutral  0
     * Red      1
     * Blue     2
     * Green    3
     * Yellow   4
     * Pink     5
     * Brown    6
     */
    //Flag Color Handling
    GameObject temp = selected.GetComponent<Province>().Flag;
    if (temp != null) {
      if (Input.GetKeyDown(KeyCode.Alpha1)) {
        selected.GetComponent<Province>().Color = "red";
        temp.GetComponent<Animator>().SetInteger("team", 1);
      }
      else if (Input.GetKeyDown(KeyCode.Alpha2)) {
        selected.GetComponent<Province>().Color = "blue";
        temp.GetComponent<Animator>().SetInteger("team", 2);
      }
      else if (Input.GetKeyDown(KeyCode.Alpha3)) {
        selected.GetComponent<Province>().Color = "green";
        temp.GetComponent<Animator>().SetInteger("team", 3);
      }
      else if (Input.GetKeyDown(KeyCode.Alpha4)) {
        selected.GetComponent<Province>().Color = "yellow";
        temp.GetComponent<Animator>().SetInteger("team", 4);
      }
      else if (Input.GetKeyDown(KeyCode.Alpha5)) {
        selected.GetComponent<Province>().Color = "pink";
        temp.GetComponent<Animator>().SetInteger("team", 5);
      }
      else if (Input.GetKeyDown(KeyCode.Alpha6)) {
        selected.GetComponent<Province>().Color = "brown";
        temp.GetComponent<Animator>().SetInteger("team", 6);
      }
      else if (Input.GetKeyDown(KeyCode.Alpha0)) {
        selected.GetComponent<Province>().Color = "neutral";
        temp.GetComponent<Animator>().SetInteger("team", 0);
      }
    }

    if (selected2 != null) {
      //Adjacency Checker
      if (Input.GetKeyDown(KeyCode.A)) {
        if (CheckAdjacency(selected.GetComponent<Province>(), selected2.GetComponent<Province>())) {
          Debug.Log(selected.GetComponent<Province>().ProvinceName + " and " + selected2.GetComponent<Province>().ProvinceName + " Are Adjacent!");
        }
        else {
          Debug.Log(selected.GetComponent<Province>().ProvinceName + " and " + selected2.GetComponent<Province>().ProvinceName + " Are Not Adjacent!");
        }
      }

      if (Input.GetKeyDown(KeyCode.D)) {
        //Attack test
        AttackSetup();
      }

      //Check for Path (for troop reinforcements)
      if (Input.GetKeyDown(KeyCode.S)) {
        OpenList.Add(selected.GetComponent<Province>());
        if (CheckForPath(selected2.GetComponent<Province>())) {
          Debug.Log("A Path From " + selected.GetComponent<Province>() + " to " + selected2.GetComponent<Province>() + " Exists!");
        }
        else {
          Debug.Log("A Path Does Not Exist!");
        }
      }

      if (Input.GetKeyDown(KeyCode.R)) {
        ReinforceSetup();
      }
    }
  }
}

public void GameSetup() {
  List<Province> provincesCopy = new List<Province>(AllProvinces);
  for(int i = 0; i < 6; i++) {
    TeamTerritories[i] = new List<Province>();
  }
  int numOfTeams = numberOfActivePlayers;
  if(numOfTeams == 2) {
    numOfTeams = 3;
  }
  while(provincesCopy.Count != 0) {
    int temp = Random.Range(0, provincesCopy.Count);
    provincesCopy[temp].Color = playerColors[currentPlayer];
    provincesCopy[temp].Flag.GetComponent<Animator>().SetInteger("team", TeamColors.IndexOf(playerColors[currentPlayer]));
```

```
      TeamTerritories[currentPlayer].Add(provincesCopy[temp]);
      provincesCopy[temp].TroopCount = 1;
      provincesCopy.RemoveAt(temp);
      if(currentPlayer == (numOfTeams - 1)) {
        currentPlayer = 0;
      } else {
        currentPlayer++;
      }
    }

    switch(numberOfActivePlayers) {
      case 2:
        numberOfTroopsToPlace = 26;
        break;
      case 3:
        numberOfTroopsToPlace = 21;
        break;
      case 4:
        numberOfTroopsToPlace = 19;
        break;
      case 5:
        numberOfTroopsToPlace = 16;
        break;
      case 6:
        numberOfTroopsToPlace = 13;
        break;
    }
    currStage = Stages.GAME_SETUP;
    currentPlayer = 0;
  }

  public void NewTurn() {
    if(selected) {
      selected.GetComponent<Province>().Deselect();
    }
    if(TeamTerritories[currentPlayer].Count == 42) {
      MenuText.text = playerColors[currentPlayer].ToUpper() + " Won the Game!";
      mainMenuVisible = true;
      mainMenu.SetActive(true);
      statusMenu.SetActive(false);
    }
    else if (!attackMenuVisible && !reinforceMenuVisible && (currStage > Stages.REPLENISH))
     ↪ {
      if (currentPlayer == (numberOfActivePlayers - 1)) {
        currentPlayer = 0;
      }
      else {
        currentPlayer++;

      }
      while(TeamTerritories[currentPlayer].Count == 0) {
        if(currentPlayer == (numberOfActivePlayers - 1)) {
          currentPlayer = 0;
        } else {
          currentPlayer++;
        }
      }
      numberOfTroopsToPlace = calculateReinforcements();
      currStage = Stages.REPLENISH;
    }
```

```
}

public bool FindInList(List<Province> subList, List<Province> superList) {
  foreach (Province province in subList) {
    if(!superList.Contains(province)) {
      return false;
    }
  }
  return true;
}

int calculateReinforcements() {
  int territoryOwnershipCount = Mathf.Max(TeamTerritories[currentPlayer].Count / 3,3);
  int continentBonus = 0;
  //Check Continental Bonuses
  if(FindInList(Europe, TeamTerritories[currentPlayer])) {
    continentBonus += 5;
  }
  if(FindInList(Asia, TeamTerritories[currentPlayer])) {
    continentBonus += 7;
  }
  if(FindInList(Africa, TeamTerritories[currentPlayer])) {
    continentBonus += 3;
  }
  if(FindInList(Oceania, TeamTerritories[currentPlayer])) {
    continentBonus += 2;
  }
  if(FindInList(SouthAmerica,TeamTerritories[currentPlayer])) {
    continentBonus += 2;
  }
  if(FindInList(NorthAmerica,TeamTerritories[currentPlayer])) {
    continentBonus += 5;
  }
  return territoryOwnershipCount + continentBonus;
}


// Update is called once per frame
void Update() {
  /* Basically I want to keep debug mode (in case of emergency) so it's activated/deactivated using
  if(Input.GetKeyDown(KeyCode.BackQuote)) {
    //Toggle Debug mode
    InDebugMode = !InDebugMode;
    Debug.Log("Debug Mode Toggled!");
  }

  if (InDebugMode) {
    DebugEval();
  } else if (!mainMenuVisible) {
    switch (currStage) {
      case Stages.NOT_STARTED:
        GameSetup();
        break;
      case Stages.GAME_SETUP:
        statusBar.text = playerColors[currentPlayer].ToUpper() + ": Place a Troop!";
        if(Input.GetMouseButtonDown(0) && !attackMenuVisible && !reinforceMenuVisible) {
          selected = GetTerritoryClick();
          if (selected &&
          ↪  selected.GetComponent<Province>().Color.Equals(playerColors[currentPlayer]))
          ↪  {
            selected.GetComponent<Province>().TroopCount++;
            if(currentPlayer == (numberOfActivePlayers - 1)) {
              currentPlayer = 0;
              numberOfTroopsToPlace--;
              if (numberOfTroopsToPlace == 0) {
                currStage = Stages.REPLENISH;
                numberOfTroopsToPlace = calculateReinforcements();
              }
            } else {
              currentPlayer++;
            }
          }
        }
        break;
      case Stages.REPLENISH:
        statusBar.text = playerColors[currentPlayer].ToUpper() + ": Place " + numberOfTroopsToPlace.ToString() + " Troops!";
        if(Input.GetMouseButtonDown(0) && !attackMenuVisible && !reinforceMenuVisible) {
          selected = GetTerritoryClick();
          if (selected && selected.GetComponent<Province>().Color.Equals(playerColors[currentPlayer])) {
            selected.GetComponent<Province>().TroopCount++;
            numberOfTroopsToPlace--;
            if(numberOfTroopsToPlace == 0) {
              currStage = Stages.ATTACK;
              selected = null;
            }
          }
        }
        break;
      case Stages.ATTACK:
        if(!selected) {
          statusBar.text = playerColors[currentPlayer].ToUpper() + ": Click a Territory to attack from";
        } else {
          statusBar.text = playerColors[currentPlayer].ToUpper() + ": Select a region to attack\nRight Click To Reinforce";
        }
        if(Input.GetMouseButtonDown(0) && !attackMenuVisible && !reinforceMenuVisible) {
          GameObject temp = GetTerritoryClick();
          if(temp) {
            if (temp.GetComponent<Province>().Color.Equals(playerColors[currentPlayer])) {
              if(selected) {
                selected.GetComponent<Province>().Deselect();
              }
              selected = temp;
              selected.GetComponent<Province>().Select();
            } else {
              selected2 = temp;
              AttackSetup();
            }
          }
        }
        if(Input.GetMouseButtonDown(1)) {
          currStage = Stages.REINFORCE;
          reinforcementPerformedThisTurn = false;
          if (selected) {
            selected.GetComponent<Province>().Deselect();
          }
          selected = null;
          selected2 = null;
        }
        break;
      case Stages.REINFORCE:
        if (!selected) {
          statusBar.text = playerColors[currentPlayer].ToUpper() + ": Select a Region to Move Troops From";
        } else {
          statusBar.text = playerColors[currentPlayer].ToUpper() + ": Select a Region to Move Troops To\n(Right Click To change departing region)";
        }
        if(Input.GetMouseButtonDown(0) && !attackMenuVisible && !reinforceMenuVisible) {
          GameObject temp = GetTerritoryClick();
          if(temp) {
            if(temp.GetComponent<Province>().Color.Equals(playerColors[currentPlayer])) {
              if(!selected) {
                temp.GetComponent<Province>().Select();
                selected = temp;
              } else {
                selected2 = temp;
                ReinforceSetup();
              }
            }
          }
        }
        if(Input.GetMouseButtonDown(1) && !attackMenuVisible && !reinforceMenuVisible) {
          GameObject temp = GetTerritoryClick();
          if(temp) {
            if(temp.GetComponent<Province>().Color.Equals(playerColors[currentPlayer])) {
              if(selected) {
                selected.GetComponent<Province>().Deselect();
              }
              selected = temp;
            }
          }
        }
        if(!reinforceMenuVisible && reinforcementPerformedThisTurn) {
          NewTurn();
        }
        break;
    }
    if(Input.GetKeyDown(KeyCode.Escape)) {
      mainMenu.SetActive(true);
      mainMenuVisible = true;
      statusMenu.SetActive(true);
    }
  }
}
}
```

6.2 Dice Roller

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

/*
 *  DiceRoller.cs
 *  Author: Daniel Hannon (19484286)
 *  Version: 1
 */

public class DiceRoller : MonoBehaviour {
  // Start is called before the first frame update
  public Sprite[] AttackDiceFaces;
  public Sprite[] DefenseDiceFaces;
  public int dice_type;
  public int current_value = 0;

  public int Roll() {
    current_value = Random.Range(1, 7);
    if (dice_type == 0) {
      //Attack Dice
      gameObject.GetComponent<Image>().sprite = AttackDiceFaces[current_value];
    }
    else {
      //Defense Dice
      gameObject.GetComponent<Image>().sprite = DefenseDiceFaces[current_value];
    }
    return current_value;
  }

  public void SetInactive() {
    gameObject.GetComponent<Image>().sprite = AttackDiceFaces[0];
  }
}
```

6.3 Province

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

/**
 *  Province.cs
 *  Author: Daniel Hannon (19484286)
 *  Version: 1
 */

public class Province : MonoBehaviour {
  public string ProvinceName;
  public string Color = "neutral";
  public int TroopCount = 0;
  public GameObject Flag;
  public GameObject TroopField;
  public GameObject[] neighbors;
  private LineRenderer highlight = null;


  void start() {
    TroopCount = 0;
    Color = "neutral";
  }

  void Update() {

    if(TroopField!= null) {
      TroopField.GetComponent<TextMesh>().text = TroopCount.ToString();
    }
  }

   public void Select() {
    //The select method creates a LineRenderer and follows the path set by the PolygonCollider used to
    //As a means of letting the user know what territory they currently have selected, I realized this
    //Some UI prompt that has something like *Currently Selected: Region Name* as region names are not
    highlight = gameObject.AddComponent<LineRenderer>();
    if (highlight) {
      Vector2[] path = gameObject.GetComponent<PolygonCollider2D>().points;
      Color black = new Color(0, 0, 0, 1);
      highlight.material = new Material(Shader.Find("Legacy Shaders/Particles/Alpha
       ↪  Blended Premultiply"));
      highlight.startColor = black;
      highlight.endColor = black;
      highlight.startWidth = 0.03f;
      for (int i = 0; i < path.Length; i++) {
        path[i] = gameObject.transform.TransformPoint(path[i]);
      }
      highlight.positionCount = path.Length + 1;
      for (int i = 0; i < path.Length; i++) {
        Vector3 finalLine = path[i];
        finalLine.z = 30;
        highlight.SetPosition(i, finalLine);

        if (i == (path.Length - 1)) {
          finalLine = path[0];
          finalLine.z = 30;
          highlight.SetPosition(path.Length, finalLine);
        }
      }
    }
  }

  public void Deselect() {
    //This destroys the LineRenderer (I was worried it'd delete the points for the PolygonCollider but
    if (highlight) {
      Destroy(gameObject.GetComponent<LineRenderer>());
      highlight = null;
    }
  }
}
```