Computer Games Development CW208
TDD
Year IV

Aeden Moylan

C00249220

24/04/2023

DECLARATION

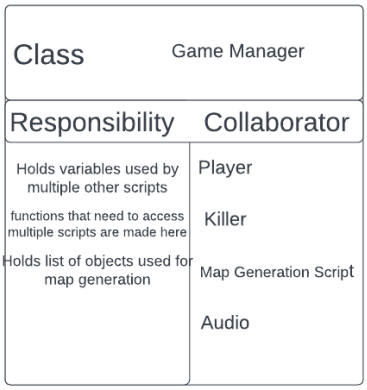
UNDERGRADUATE ASSESSMENT WORK DECLARATION

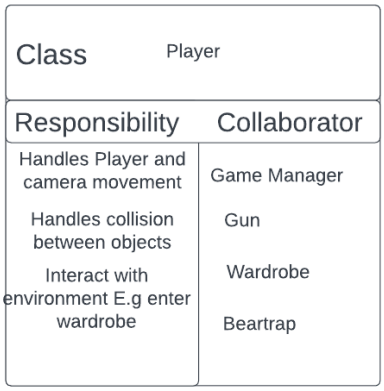
Work submitted for assessment which does not include this declaration will not be assessed.

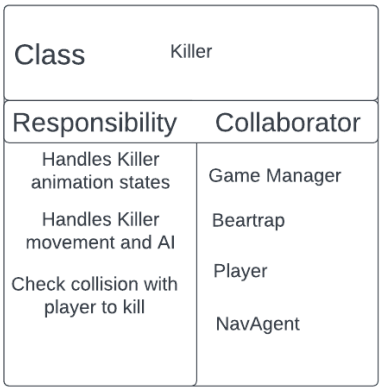
- I declare that all material in this submission e.g. thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.
- I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams or other material; including software and other electronic media in which intellectual property rights may reside.
- I have provided a complete bibliography of all works and sources used in the preparation of this submission.
- I understand that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offence.

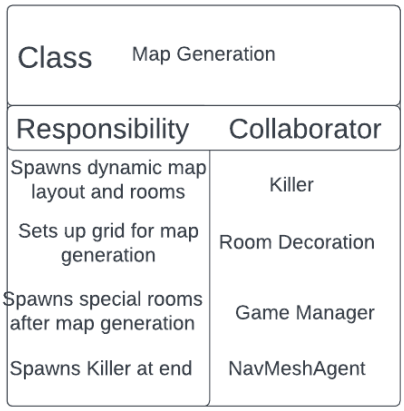
Student Name: (Printed)	AEDEN MOYLAN
Student Number(s):	C00249220
Signature(s):	Aeden Moylan
Date:	24/04/2023

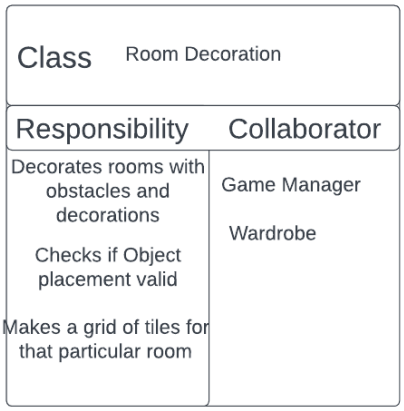
CRC Cards







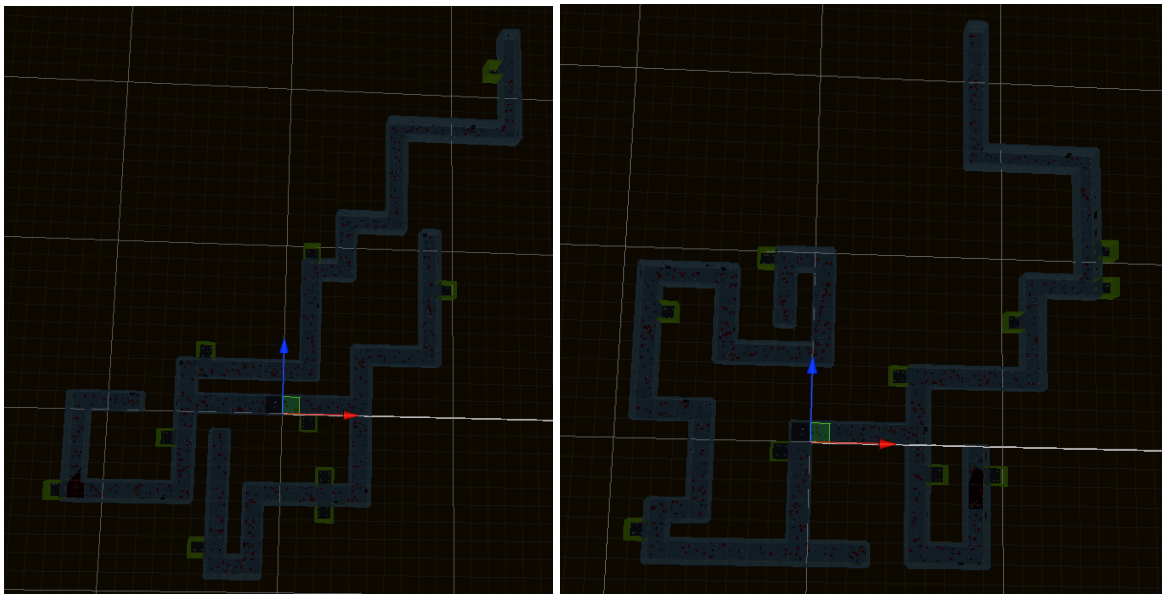




Algorithm explanation

Map generation algorithm:

The map generation algorithm is used to make a random map layout. The results are entirely random and the map can look like the examples shown below



I will explain the steps I took to achieve this:

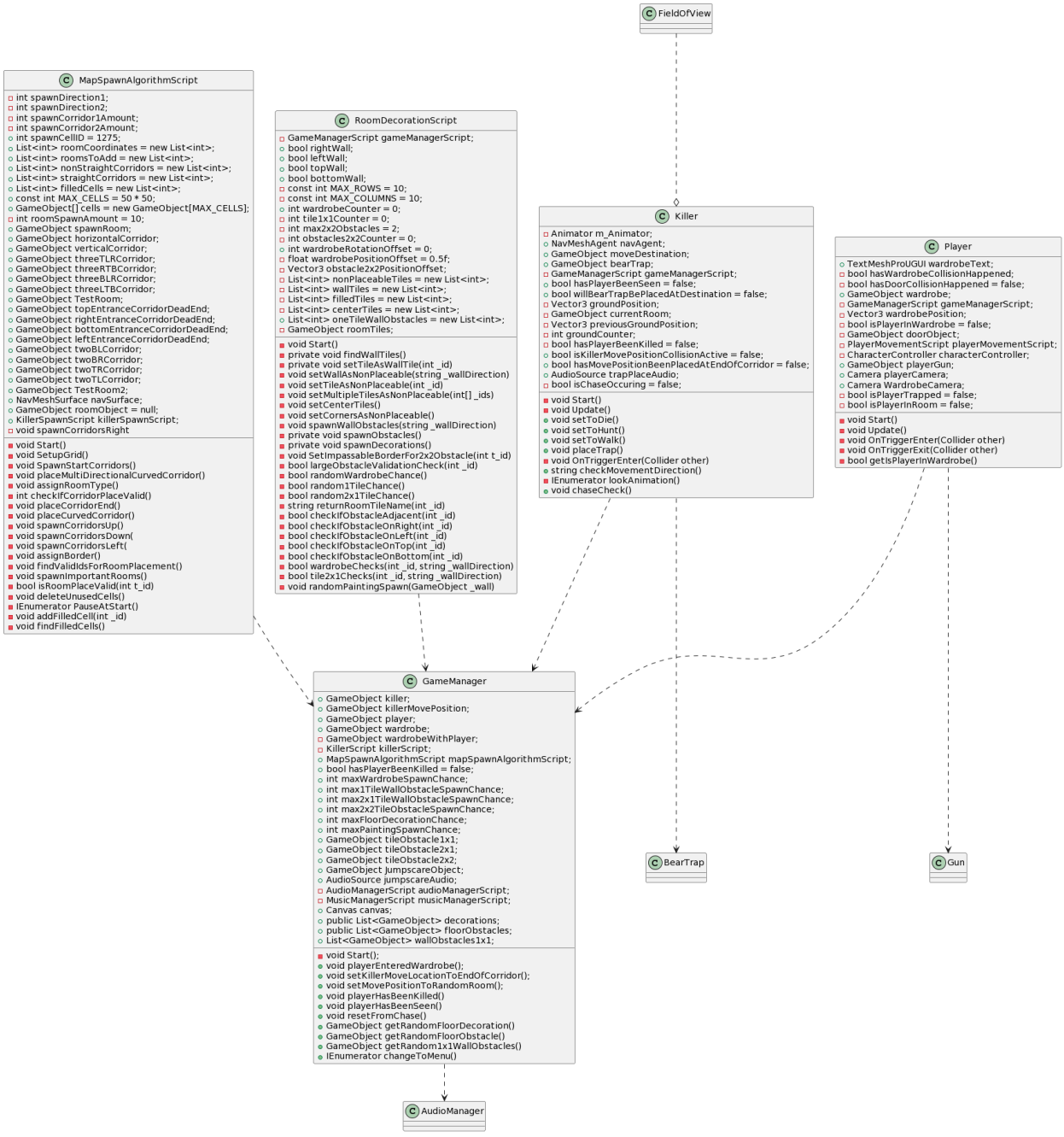
First I made a 50 x 50 grid of cells. Each cell represents a potential space for a room to be placed. This means that there are 2500 potential spots for a room. A spawn room is placed in the middle and assigned an id of 1249. The spawn room then picks 2 random directions for corridors to be built from. The walls of the spawn room are broken in this direction and corridors are placed. After a random amount of corridors have been placed, a multi-curved corridor then gets placed. This allows corridors to be spawned in 2 directions. Once the end of the corridors has been met, the algorithm places curved corridors which start making corridors on it. This loop continues until a room placement isn't valid. If this happens an end corridor is placed to ensure no more corridors will be built. When a room is placed, information about the room is stored in the cell id from the grid that is associated with the room. This includes what type of room it is and other relevant information. This is used for multiple checks for different areas including room placement validation. These validation checks if a corridor is about to be built into an already existing room or the border of the grid. After the basic map is made, special rooms then get inserted around the map. These can only be inserted adjacent to corridors and if the special room location has only 1 neighbour. Once a location is deemed valid, the id of the cell associated with it is stored in a list. This continues until all valid placements have been gathered. After this, random cell Ids are picked from this list and rooms are placed in that location.

Room Decoration

The floor in each room is split up into a 10 x 10 set of tiles. Each tile represents a potential area where obstacles can be placed. Each tile will be split up into categories. These are wall tiles, Center tiles, Filled tiles, non-placeable tiles and available tiles. Wall tiles are tiles that are closest to the wall of the room. Only certain types of obstacles can be placed here as they depend on a wall. Center Tiles are tiles that are not close to a wall. These are in the centre of the room and

big impassable obstacles, as well as floor decorations, will be placed here. Filled tiles are tiles that already have an obstacle or decoration in that area. No more obstacles can be placed here but decorations can be placed as they do not block the path. Available tiles are the remaining spaces where obstacles can be placed. Obstacles cannot be placed on non-placeable tiles. If a room has a special room attached to it, the tiles facing the wall that the room is touching are instantly turned into non-placeable tiles to not conflict with accessing the door. The if there is a wall, wall tiles are generated and the assets are placed. When an asset is placed, the border tiles around it are turned into non-passable tiles. This is to ensure that the path is still traversable. These wall assets have different sizes and some even take up multiple tiles each. Many checks are done to assure that the appropriate tiles are filled in. Among these obstacles are wardrobes which the player can hide in. These have a chance to be spawned first as they act as a core game mechanic. After this, the centre tiles are given a chance to spawn obstacles. The obstacles spawned in the centre area take up a 2x2 tile area. Multiple checks are needed to make sure placement is valid and when it is placed, the border is set to non-passable just like the wall objects. This is especially important here as the path needs to be traversable. Once the obstacles are placed, the decorations can be placed in the centre tile areas even if an obstacle is in that same spot. The size of the decorations does not matter as much as they do not have any collisions and are purely aesthetic, however, I used thin objects that would easily be flat on the floor such as paper or a blood stain.

Class Diagram


FieldOfView
MapSpawnAlgorithmScript
- int spawnDirection1;
- int spawnDirection2;
- int spawnCorridor1Amount;
- int spawnCorridor2Amount;
+ int spawnCellID = 1275;
+ List<int> roomCoordinates = new List<int>;
+ List<int> roomsToAdd = new List<int>;
+ List<int> nonStraightCorridors = new List<int>;
+ List<int> straightCorridors = new List<int>;
+ List<int> filledCells = new List<int>;
+ const int MAX_CELLS = 50 * 50;
+ GameObject[] cells = new GameObject[MAX_CELLS];
- int roomSpawnAmount = 10;
+ GameObject spawnRoom;
+ GameObject horizontalCorridor;
+ GameObject verticalCorridor;
+ GameObject threeTLRCorridor;
+ GameObject threeRTBCorridor;
+ GameObject threeBLRCorridor;
+ GameObject threeLTBCorridor;
+ GameObject TestRoom;
+ GameObject topEntranceCorridorDeadEnd;
+ GameObject rightEntranceCorridorDeadEnd;
+ GameObject bottomEntranceCorridorDeadEnd;
+ GameObject leftEntranceCorridorDeadEnd;
+ GameObject twoBLCorridor;
+ GameObject twoBRCorridor;
+ GameObject twoTRCorridor;
+ GameObject twoTLCorridor;
+ GameObject TestRoom2;
+ NavMeshSurface navSurface;
+ GameObject roomObject = null;
+ KillerSpawnScript killerSpawnScript;
- void spawnCorridorsRight
- void Start()
- void SetupGrid()
- void SpawnStartCorridors()
- void placeMultiDirectionalCurvedCorridor()
- void assignRoomType()
- int checkIfCorridorPlaceValid()
- void placeCorridorEnd()
- void placeCurvedCorridor()
- void spawnCorridorsUp()
- void spawnCorridorsDown(
- void spawnCorridorsLeft(
- void assignBorder()
- void findValidIdsForRoomPlacement()
- void spawnImportantRooms()
- bool isRoomPlaceValid(int t_id)
- void deleteUnusedCells()
- IEnumerator PauseAtStart()
- void addFilledCell(int _id)
- void findFilledCells()
RoomDecorationScript
- GameManagerScript gameManagerScript;
+ bool rightWall;
+ bool leftWall;
+ bool topWall;
+ bool bottomWall;
- const int MAX_ROWS = 10;
- const int MAX_COLUMNS = 10;
+ int wardrobeCounter = 0;
- int tile1x1Counter = 0;
- int max2x2Obstacles = 2;
- int obstacles2x2Counter = 0;
+ int wardrobeRotationOffset = 0;
- float wardrobePositionOffset = 0.5f;
- Vector3 obstacle2x2PositionOffset;
- List<int> nonPlaceableTiles = new List<int>;
- List<int> wallTiles = new List<int>;
- List<int> filledTiles = new List<int>;
- List<int> centerTiles = new List<int>;
+ List<int> oneTileWallObstacles = new List<int>;
- GameObject roomTiles;
- void Start()
- private void findWallTiles()
- private void setTileAsWallTile(int _id)
- void setWallAsNonPlaceable(string _wallDirection)
- void setTileAsNonPlaceable(int _id)
- void setMultipleTilesAsNonPlaceable(int[] _ids)
- void setCenterTiles()
- void setCornersAsNonPlaceable()
- void spawnWallObstacles(string _wallDirection)
- private void spawnObstacles()
- private void spawnDecorations()
- void SetImpassableBorderFor2x2Obstacle(int t_id)
- bool largeObstacleValidationCheck(int _id)
- bool randomWardrobeChance()
- bool random1TileChance()
- bool random2x1TileChance()
- string returnRoomTileName(int _id)
- bool checkIfObstacleAdjacent(int _id)
- bool checkIfObstacleOnRight(int _id)
- bool checkIfObstacleOnLeft(int _id)
- bool checkIfObstacleOnTop(int _id)
- bool checkIfObstacleOnBottom(int _id)
- bool wardrobeChecks(int _id, string _wallDirection)
- bool tile2x1Checks(int _id, string _wallDirection)
- void randomPaintingSpawn(GameObject _wall)
Killer
- Animator m_Animator;
+ NavMeshAgent navAgent;
+ GameObject moveDestination;
+ GameObject bearTrap;
- GameManagerScript gameManagerScript;
+ bool hasPlayerBeenSeen = false;
+ bool willBearTrapBePlacedAtDestination = false;
- Vector3 groundPosition;
- GameObject currentRoom;
- Vector3 previousGroundPosition;
- int groundCounter;
- bool hasPlayerBeenKilled = false;
+ bool isKillerMovePositionCollisionActive = false;
+ bool hasMovePositionBeenPlacedAtEndOfCorridor = false;
+ AudioSource trapPlaceAudio;
- bool isChaseOccuring = false;
- void Start()
- void Update()
+ void setToDie()
+ void setToHunt()
+ void setToWalk()
+ void placeTrap()
- void OnTriggerEnter(Collider other)
+ string checkMovementDirection()
- IEnumerator lookAnimation()
+ void chaseCheck()
Player
+ TextMeshProUGUI wardrobeText;
- bool hasWardrobeCollisionHappened;
- bool hasDoorCollisionHappened = false;
+ GameObject wardrobe;
- GameManagerScript gameManagerScript;
- Vector3 wardrobePosition;
- bool isPlayerInWardrobe = false;
- GameObject doorObject;
- PlayerMovementScript playerMovementScript;
- CharacterController characterController;
+ GameObject playerGun;
+ Camera playerCamera;
+ Camera WardrobeCamera;
- bool isPlayerTrapped = false;
- bool isPlayerInRoom = false;
- void Start()
- void Update()
- void OnTriggerEnter(Collider other)
- void OnTriggerExit(Collider other)
- bool getIsPlayerInWardrobe()
GameManager
+ GameObject killer;
+ GameObject killerMovePosition;
+ GameObject player;
+ GameObject wardrobe;
- GameObject wardrobeWithPlayer;
- KillerScript killerScript;
+ MapSpawnAlgorithmScript mapSpawnAlgorithmScript;
+ bool hasPlayerBeenKilled = false;
+ int maxWardrobeSpawnChance;
+ int max1TileWallObstacleSpawnChance;
+ int max2x1TileWallObstacleSpawnChance;
+ int max2x2TileObstacleSpawnChance;
+ int maxFloorDecorationChance;
+ int maxPaintingSpawnChance;
+ GameObject tileObstacle1x1;
+ GameObject tileObstacle2x1;
+ GameObject tileObstacle2x2;
+ GameObject JumpscareObject;
+ AudioSource jumpscareAudio;
- AudioManagerScript audioManagerScript;
- MusicManagerScript musicManagerScript;
+ Canvas canvas;
+ public List<GameObject> decorations;
+ public List<GameObject> floorObstacles;
+ List<GameObject> wallObstacles1x1;
- void Start();
+ void playerEnteredWardrobe();
+ void setKillerMoveLocationToEndOfCorridor();
+ void setMovePositionToRandomRoom();
+ void playerHasBeenKilled()
+ void playerHasBeenSeen()
+ void resetFromChase()
+ GameObject getRandomFloorDecoration()
+ GameObject getRandomFloorObstacle()
+ GameObject getRandom1x1WallObstacles()
+ IEnumerator changeToMenu()
BearTrap
Gun
AudioManager