

Unity's Pac-Man

Titolo del progetto: Unity's Pac-Man
Alunno/a: Michael Dobeson & Luca Fumasoli
Classe: Info 3AC
Anno scolastico: 2022/2023
Docente responsabile: Guido Montalbetti

Sommario

Sommario	2
Indice delle Figure	3
1 Introduzione	4
1.1 Informazioni sul progetto	4
1.2 Abstract	4
1.3 Scopo	4
2 Analisi	5
2.1 Analisi del dominio	5
2.2 Analisi e specifica dei requisiti	5
2.3 Use case	9
2.4 Pianificazione	10
2.5 Analisi dei mezzi	11
2.5.1 Software	11
2.5.2 Hardware	11
3 Progettazione	12
3.1 Design dell'architettura del sistema	12
3.2 Design delle interfacce	13
3.2.1 Interfaccia pagina home	13
3.2.2 Interfaccia pagina Opzioni	13
3.2.3 Interfaccia pagina di gioco	14
3.3 Design procedurale	15
3.3.1 Iniziale	15
3.3.2 Finale	17
4 Implementazione	19
4.1 Scripts	19
4.1.1 Grid Manager	19
4.1.2 Pac-Man Manager	26
4.1.3 Enemy Manager	34
4.1.4 Pill Manager	39
4.1.5 Game Mode Manager	41
4.1.6 Settings Manager	43
4.1.7 Menu Manager	47
4.2 Unity Engine	49
4.2.1 Impostazioni	49
4.2.2 Scene Manager	55
4.2.3 Prefab	56
4.2.4 Animazioni	58
4.2.5 In-Game GUI	64
4.2.6 Build and Run	66
5 Test	68
5.1 Protocollo di test	68
5.2 Risultati test	75
5.3 Mancanze/limitazioni conosciute	78
6 Consuntivo	79
7 Conclusioni	80
7.1 Sviluppi futuri	80
7.2 Considerazioni personali	80
7.2.1 Michael Dobeson	80
7.2.2 Luca Fumasoli	80
8 Glossario	81
9 Bibliografia	82
9.1 Sitografia	82

Indice delle Figure

Figura 1 - Use Case.....	9
Figura 2 - Gantt.....	10
Figura 3 - Design dell'architettura del sistema	12
Figura 4 – Pagina di home	13
Figura 5 - Pagina settaggi.....	13
Figura 6 - Interfaccia pagina di gioco	14
Figura 7 - Diagramma delle classi	15
Figura 8 - Diagramma delle classi finale	17
Figura 9 - Manhattan Distance	24
Figura 10 - Manhattan Mapper	24
Figura 11 - Unity Preferences.....	49
Figura 12 - Unity Preferences External Tools.....	50
Figura 13 - Unity Project Settings	51
Figura 14 - Unity Script Execution Order	52
Figura 15 - Unity Input Manager	53
Figura 16 - Unity Input Manager Details	54
Figura 25 - SceneManager	55
Figura 26 - Crea Prefab	56
Figura 27 - Aggiungi Canvas	56
Figura 28 - Aggiungi SpriteRenderer	57
Figura 29 - Animator ed Animation	57
Figura 17 - Unity Create Animator Controller	58
Figura 18 - Unity Animator Idle	59
Figura 19 - Unity Animator State	59
Figura 20 - Unity Animator Make Transition	59
Figura 21 - Unity Animator Transition	59
Figura 22 - Unity Animator Parameters	60
Figura 23 - Unity Animator Transition True	60
Figura 24 - Unity Create Animation	61
Figura 25 - Unity Animation Connect.....	62
Figura 26 - Unity Animation Prefab	62
Figura 27 - Unity Animation Timeline	63
Figura 28 - Unity Animator Controller Final	63
Figura 30 - Unity UI.....	64
Figura 31 - Rect Transform.....	64
Figura 32 - TextMeshPro	65
Figura 33 - Graphic Raycaster	65
Figura 29 - Unity Find Build Settings	66
Figura 30 - Unity Build Settings	66
Figura 31 - Unity Applicazione Finale	67
Figura 32 - Mappa generata	75
Figura 33 - Manga pillole	75
Figura 34 - Fantasma vittimizzato	76
Figura 35 - Incremento punteggio	76
Figura 36 - Generazione nuova mappa.....	76
Figura 37 - Game Over	77
Figura 38 - Pausa	77

1 Introduzione

1.1 Informazioni sul progetto

Allievi: Michael Dobeson, Luca Fumasoli
 Classe: I3AC
 Docente responsabile: Guido Montalbetti
 Data inizio: 09.09.2022
 Data fine: 23.12.2022

1.2 Abstract

Pac-Man is a well-known videogame which has stood the test of time and cemented itself has a classic. In this game the player has to control Pac-Man in a maze that has a pill on every cell and the players goal is to eat all the pills. In the meantime, there are 4 ghosts following the player and if they catch you, you lose a life. If the player loses al his lives the game is over. There are also super-pills in the maze which make Pac-Man invincible and capable of eating the ghosts. In our variation of Pac-Man there is only one ghost and the maze is randomly generated each game.

1.3 Scopo

Lo scopo di questo progetto è di avere una variazione di Pac-Man con 1 fantasma e dei labirinti randomizzati per continuare a rendere le partite diverse ed eliminare la familiarità che il giocatore potrebbe avere con la mappa forzandolo ad improvvisare ogni partita.

2 Analisi

2.1 Analisi del dominio

Il gioco dovrebbe essere il più semplice possibile da utilizzare dall'utente, esattamente come il gioco originale. Gli utenti dovrebbero essere già familiari con il gioco originale e giocando alla nostra versione dovrebbero capire velocemente come funziona.

2.2 Analisi e specifica dei requisiti

Priorità 2 = opzionale

ID: REQ-01	
Nome	Generazione Mappa
Priorità	1
Versione	1.0
Note	Si deve usare l'algoritmo Manhattan Mapper per generare una mappa.
Sotto requisiti	
001	Conoscenza base di Unity.

ID: REQ-02	
Nome	Movimento Pac-Man
Priorità	1
Versione	1.0
Note	Il giocatore può usare la tastiera per muoversi dentro la mappa.
Sotto requisiti	
001	Una matrice su cui può muoversi.

ID: REQ-03	
Nome	Piazzamento Pillole
Priorità	1
Versione	1.0
Note	Vengono piazzate le pillole e Super-Pillole, che aumentano il punteggio, in modo casuale dentro la mappa.
Sotto requisiti	
001	Una matrice su cui piazzarle.

ID: REQ-04	
Nome	AI Fantasma
Priorità	1
Versione	1.0
Note	Usando l'algoritmo Manhattan Mapper l'AI viene piazzato sulla mappa per inseguire il giocatore.
Sotto requisiti	
001	Una matrice su cui può muoversi.
002	La posizione del giocatore.

ID: REQ-05	
Nome	Super-Pillole
Priorità	1
Versione	1.0
Note	Quando Pac-Man consuma una Super-Pillola il fantasma scappa (Manhattan inverso) e se viene raggiunto inizia a fuggire più velocemente per un tempo parametrizzabile per poi rinascere e ritornare ad inseguire Pac-Man.
Sotto requisiti	
001	Le pillole sono piazzate.
002	Il giocatore riesce a consumarle.
003	Un'AI che segue il giocatore.

ID: REQ-06	
Nome	Grafica
Priorità	1
Versione	1.0
Note	Un'interfaccia con i bottoni per cominciare, uscire e ricominciare. Dettagli del gioco come le vite, il punteggio e l'Highscore che viene salvato in una variabile globale.
Sotto requisiti	
001	Le pillole sono piazzate.
002	Il giocatore riesce a consumarle.
003	Un'AI che segue il giocatore.

ID: REQ-07	
Nome	Vittoria/Perdita/Ricomincia
Priorità	1
Versione	1.0
Note	<p>I diversi casi in cui una nuova mappa viene generata ed i personaggi vengono ripristinati:</p> <ul style="list-style-type: none"> • Quando tutte le pillole e Super-Pillole sono consumate (Vittoria). • Quando perdi tutte le tue vite per colpa del fantasma che ti colpisce (Perdita). • Quando metti in pausa il gioco e clicchi ricomincia.
Sotto requisiti	
001	Le pillole sono piazzate.
002	Il giocatore riesce a consumarle.
003	Un'AI che segue il giocatore.
004	Pulsanti per ricominciare la partita.

ID: REQ-08	
Nome	Variabili del Gioco
Priorità	2
Versione	1.0
Note	Prima di iniziare la partita, si potrà modificare delle variabili di gioco (Vite, tempo Super-Pillole, grandezza delle mappe, velocità del fantasma, ecc..).
Sotto requisiti	
001	Il gioco finito e funzionante.

ID: REQ-09	
Nome	Difficoltà
Priorità	2
Versione	1.0
Note	Si potrà scegliere il livello di difficoltà prima di iniziare la partita.
Sotto requisiti	
001	Il gioco finito e funzionante.
002	Si può cambiare le variabili del gioco facilmente.

ID: REQ-10	
Nome	Gaming controller
Priorità	3
Versione	1.0
Note	Si potrà navigare il menu e giocare al gioco usando solo il controller.
Sotto requisiti	
001	Il gioco finito e funzionante.
002	Il menu finito e funzionante.

2.3 Use case

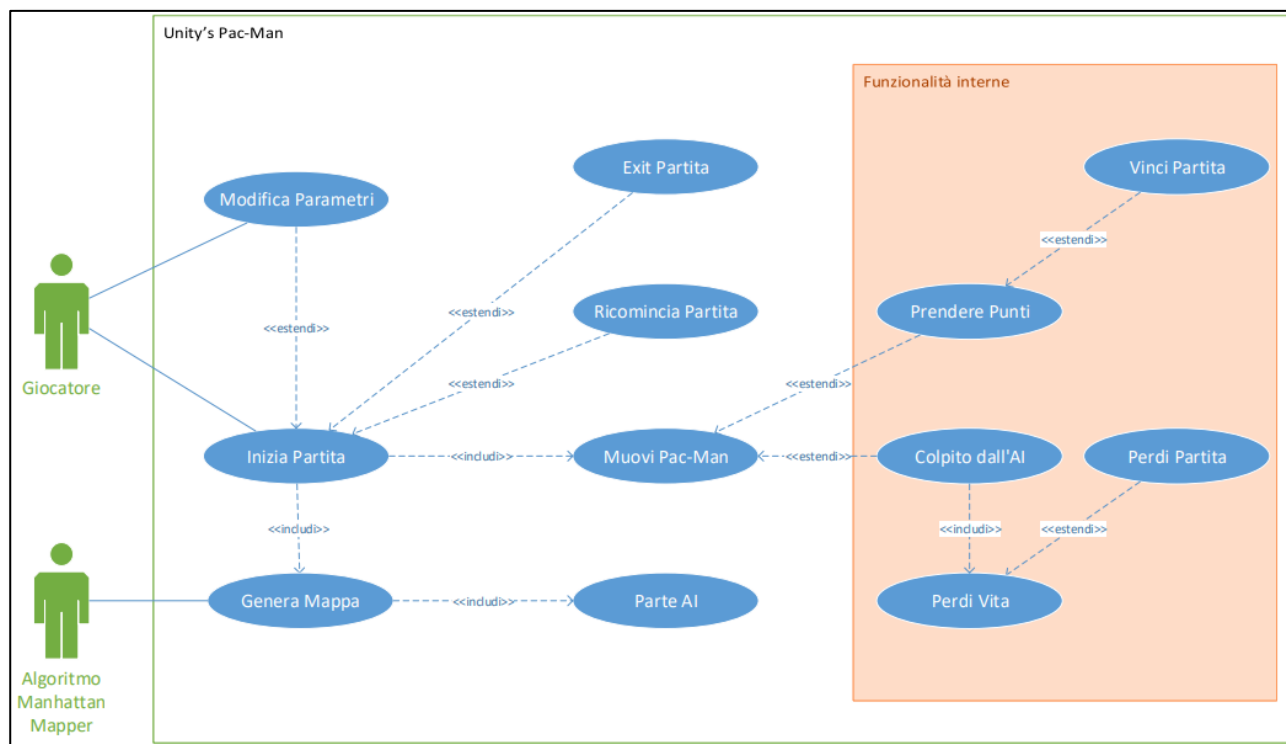


Figura 1 - Use Case

Ci sono 2 tipi di utente:

1. **Giocatore:** Il giocatore può modificare i parametri del gioco a suo piacimento prima di iniziare la partita. Poi quando avvia la partita muove il Pac-Man per il labirinto. Mangiando le pillole nel labirinto guadagna punti, se li mangia tutti vince la partita. Se viene colpito dal fantasma perde una vita e alla perdita di tutte le sue vite perde la partita. Può anche mettere in pausa la partita e ricominciarla o uscire.
2. **Algoritmo Manhattan Mapper:** Si occupa di generare la mappa randomica e muovere l'AI, usando l'algoritmo trova la strada più corta per arrivare al giocatore e la segue, se invece il giocatore ha consumato una super-pillola trova la strada più efficace per scappare dal giocatore.

2.4 Pianificazione

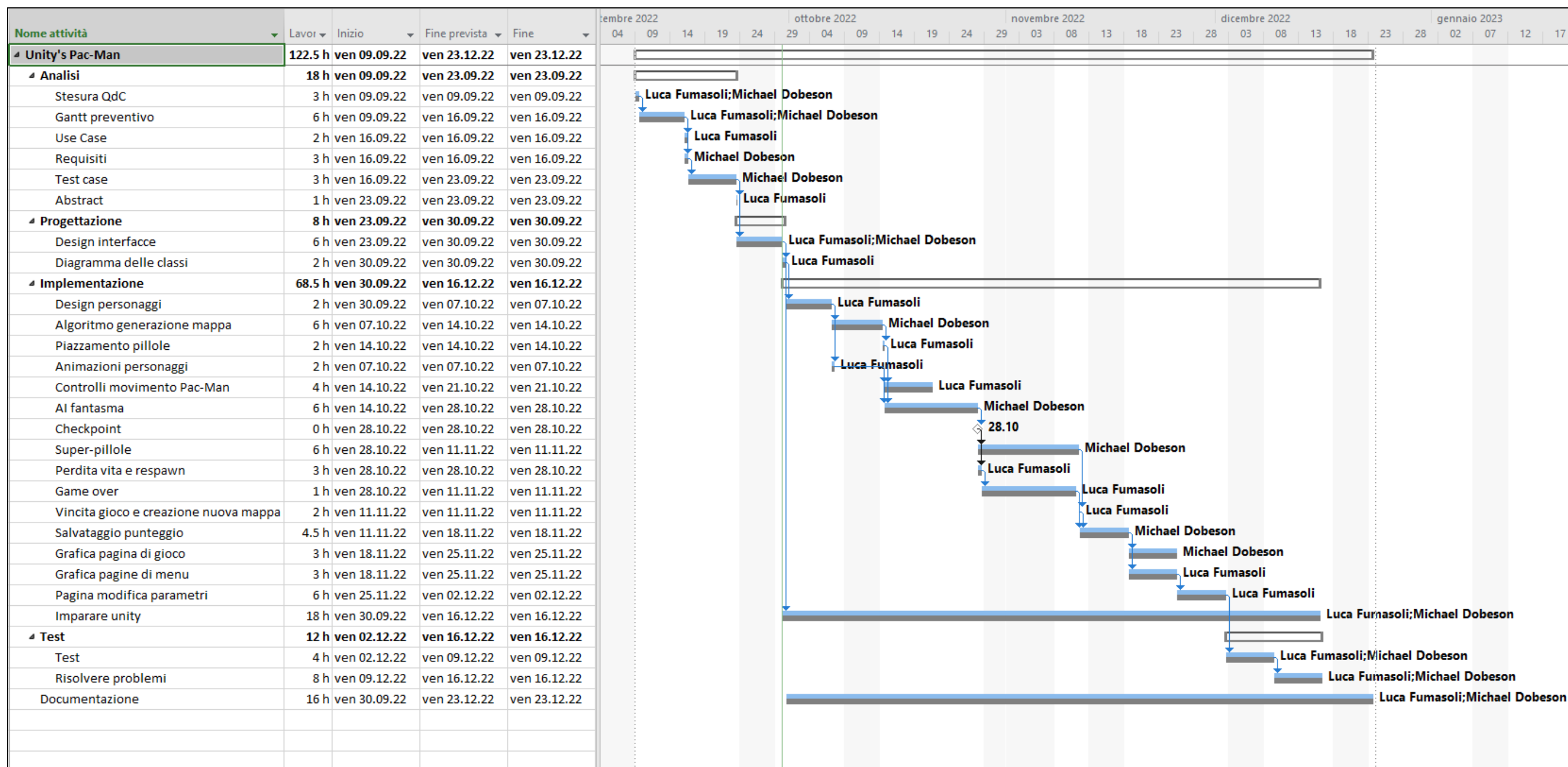


Figura 2 - Gantt

2.5 Analisi dei mezzi

2.5.1 Software

I software che abbiamo usato sono:

- Microsoft Office Professional Plus 2019
- Microsoft Word: per fare la documentazione
- Microsoft Visio: per fare lo schema del sito e dei database
- Microsoft Project: per fare il diagramma di Gantt
- PlantUML: Per fare il diagramma di flusso e di classi
- Paint versione 10.0: per il design delle interfacce
- Unity 2022.1.1f1

2.5.2 Hardware

Come hardware abbiamo usato i computer scolastici:

- Processore Intel Core i7-9700 CPU @ 3.00GHz
- RAM: 32GB
- Scheda video: Intel(R) UHD Graphics 630

3 Progettazione

3.1 Design dell'architettura del sistema

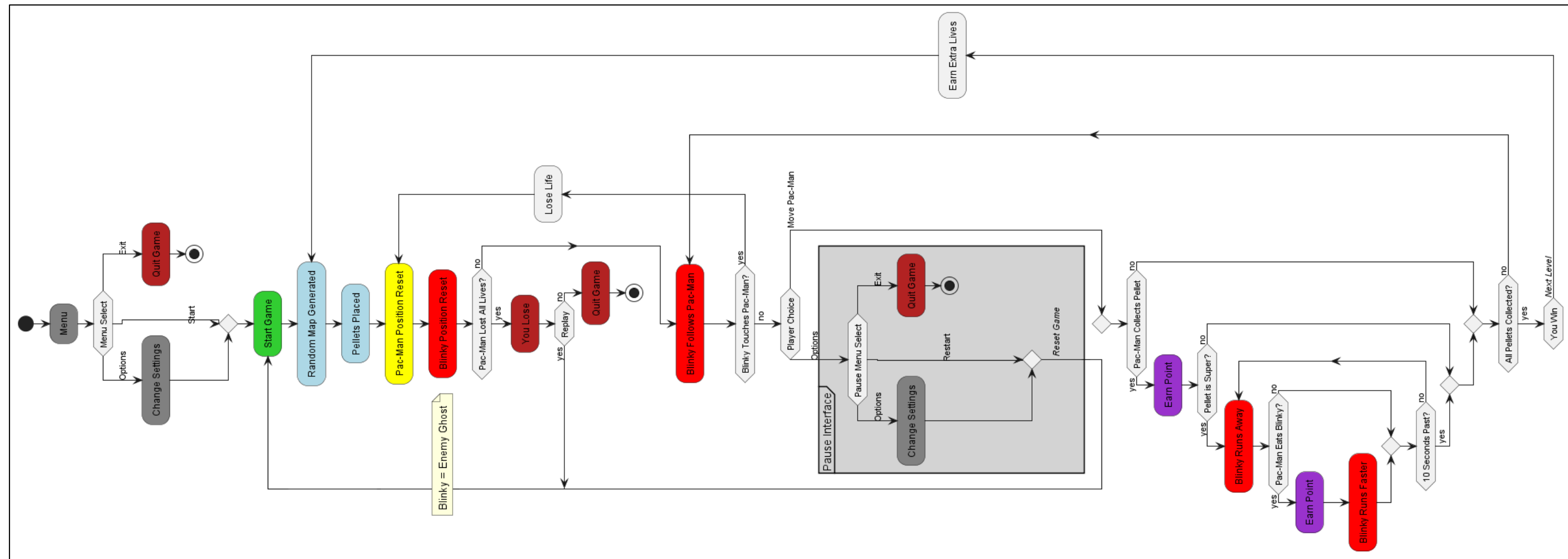


Figura 3 - Design dell'architettura del sistema

3.2 Design delle interfacce

3.2.1 Interfaccia pagina home

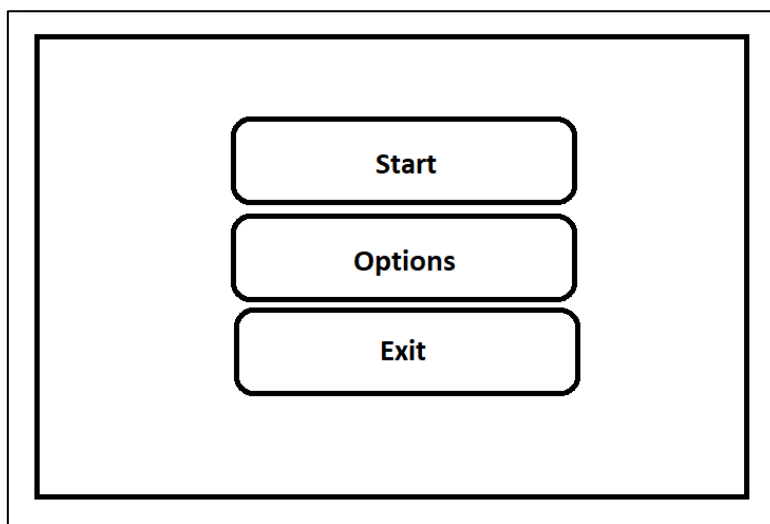


Figura 4 – Pagina di home

Bottoni per iniziare la partita, andare alla pagina delle opzioni od uscire dal gioco.

3.2.2 Interfaccia pagina Opzioni

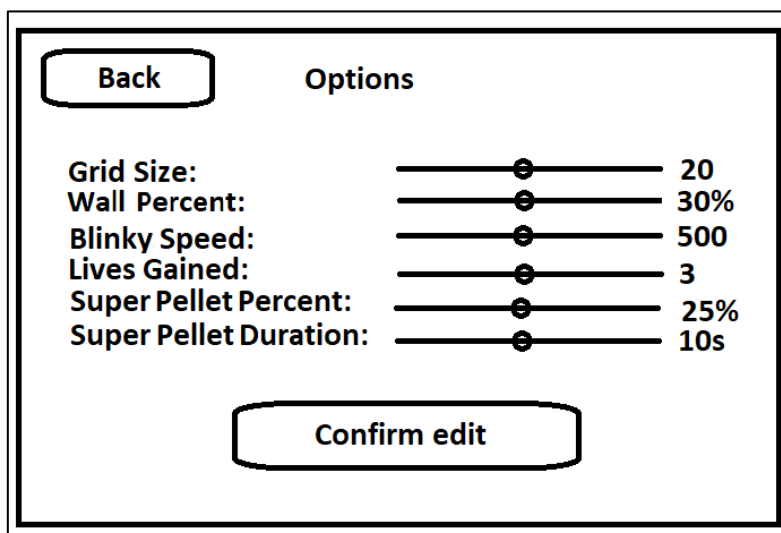


Figura 5 - Pagina settaggi

Pagina opzioni in cui si possono modificare le variabili di gioco.

3.2.3 Interfaccia pagina di gioco

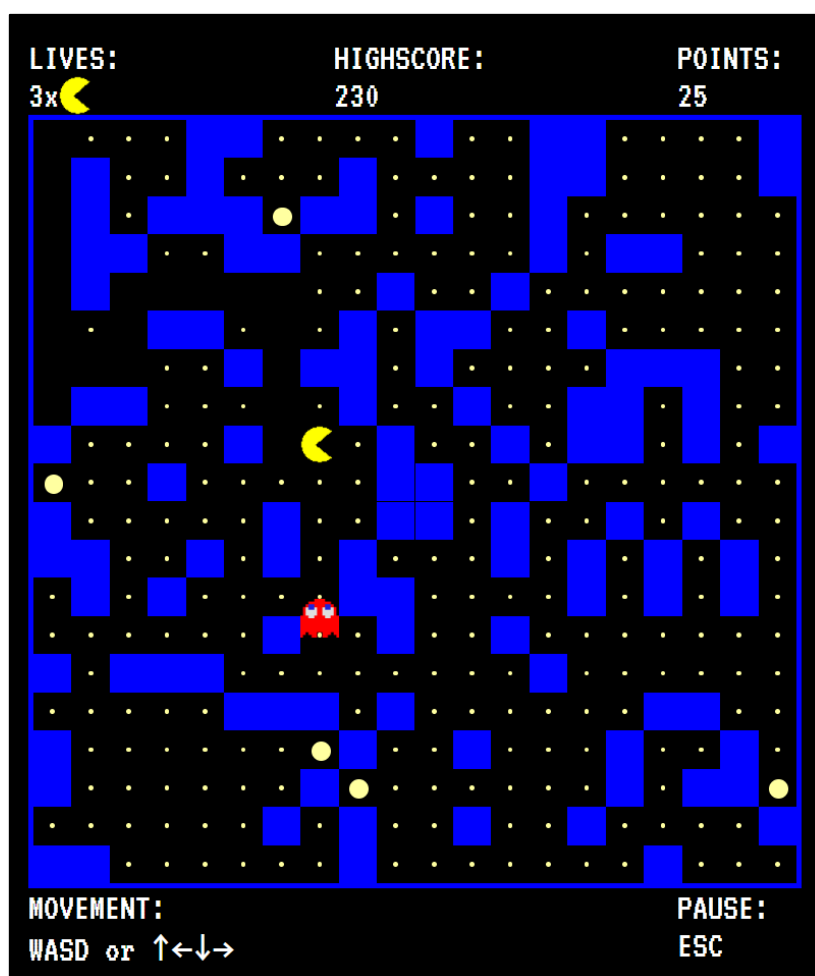


Figura 6 - Interfaccia pagina di gioco

Pagina di gioco con il labirinto ed il personaggio da muovere in centro ed i comandi sotto. In cima alla pagina a sinistra ci sono il numero di vite rimaste, in centro c'è il miglior punteggio ottenuto, ed a destra il punteggio corrente della partita.

3.3 Design procedurale

3.3.1 Iniziale

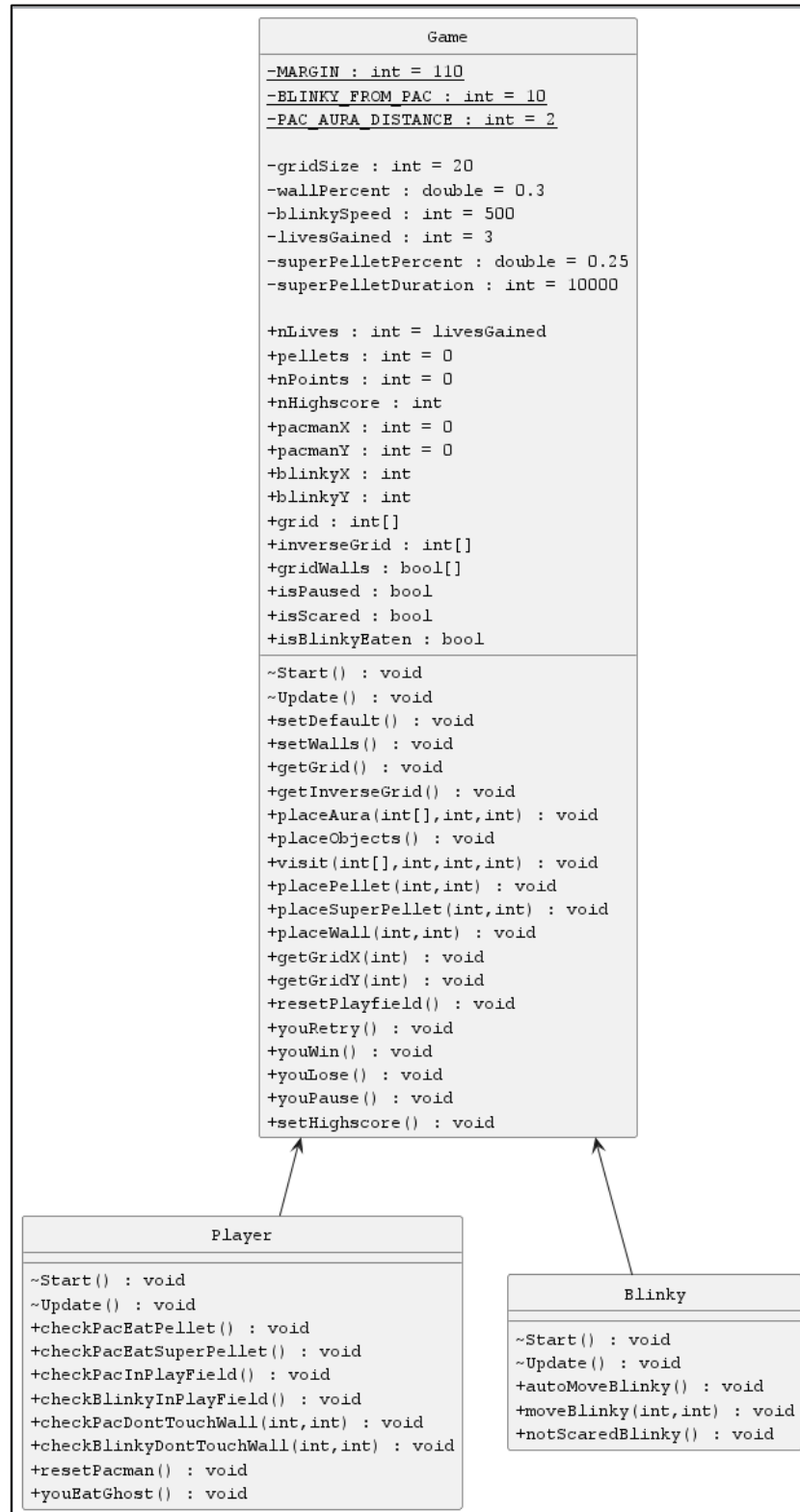


Figura 7 - Diagramma delle classi

Ci sono 3 classi:

1. **Game:** Contiene tutti i parametri modificabili del gioco e decrementa le vite quando il giocatore viene preso.
Genera il labirinto e ci inserisce dentro le pillole e super-pillole ed incrementa il punteggio.
Resetta i personaggi quando il giocatore od il fantasma viene preso.
Gestisce il menu di pausa permettendo al giocatore di uscire o resettare la partita.
Controlla che il giocatore vinca, e quindi genera una nuova mappa aumentando vite e punteggio, o perde la partita e quindi manda il giocatore alla pagina di game over.
2. **Player:** Ha dentro i metodi per far mangiare le pillole e le super pillole dal giocatore ed i metodi per controllare che i personaggi rimangano dentro il labirinto quando si muovono.
3. **Blinky:** Ha dentro i metodi per muovere il fantasma AI per il labirinto verso il giocatore o farlo scappare dal giocatore a dipendenza del valore di un booleano.

3.3.2 Finale

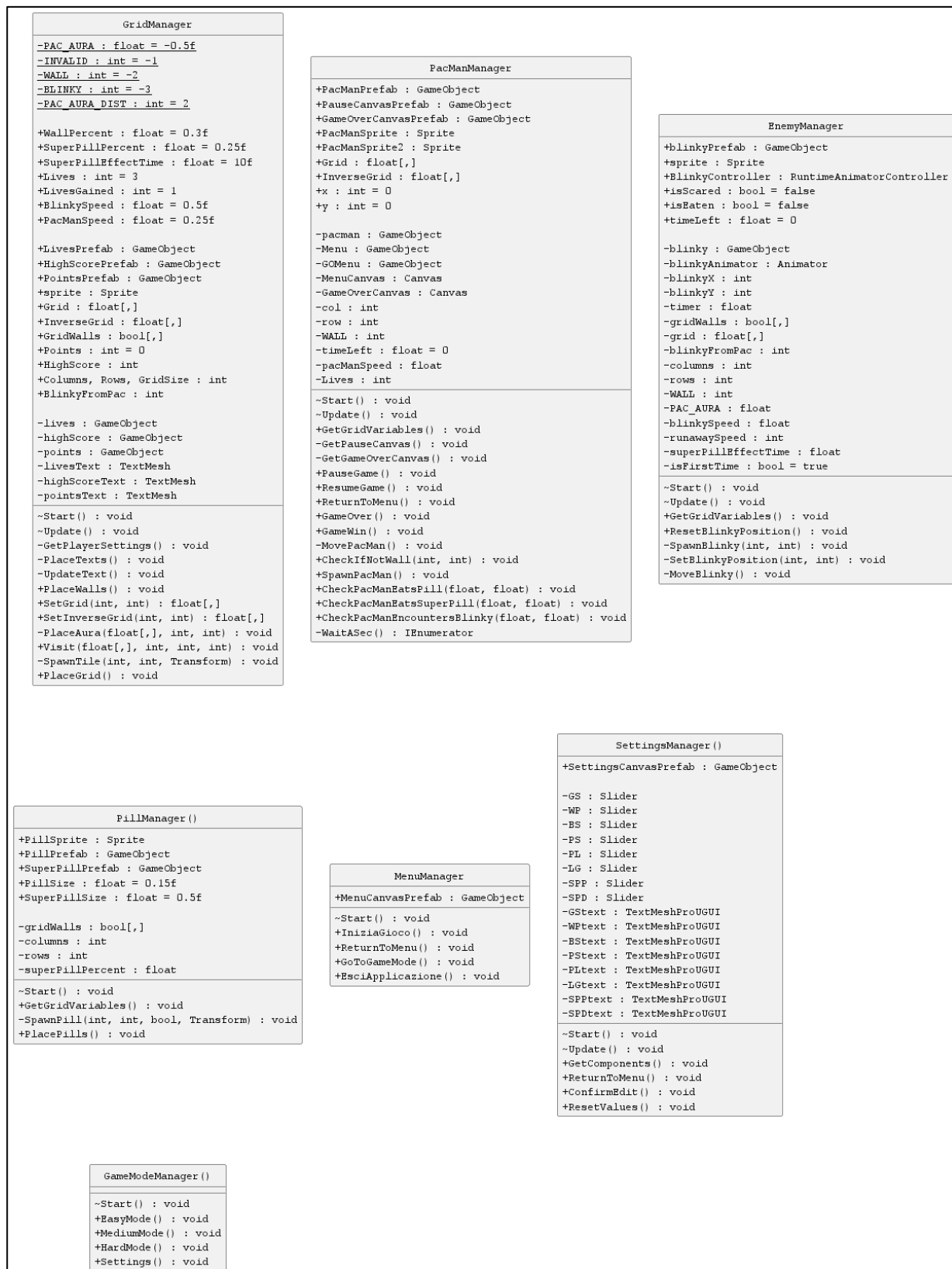


Figura 8 - Diagramma delle classi finale

Nel diagramma finale le classi sono cambiate e sono aumentate.

Classi:

1. **GridManager:** Contiene tutte le variabili modificabili del gioco per le durate degli effetti e la quantità dei componenti sulla griglia. Contiene le variabili per le *prefab*, i punteggi, le vite, le mura della griglia.
Ha dentro i metodi per poter genera la griglia con tutte le mura.
2. **PacManManager:** Contiene le variabili per le coordinate di *Pac-Man*, ha le variabili per le *prefab* ed i *canvas* di *GameOver* e pausa, il tempo di attesa per muoversi. Prende dal *GridManager* la griglia e le vite.
Ha dentro i metodi per far mangiare le pillole e le super pillole, per muovere *Pac-Man* ed assicurarsi che non esce dal labirinto o vada contro delle mura.
Gestisce il menu di pausa permettendo al giocatore di uscire o resettare la partita.
Controlla che il giocatore vinca ed aumenta le vite ed il punteggio, o se perde la partita manda il giocatore alla pagina di *game over*.
Resetta i personaggi quando il giocatore viene preso.
3. **EnemyManager:** Contiene le variabili per le coordinate del fantasma, la sprite, 2 boolean per sapere in che stato è, il tempo per cui rimane nello stato vittimizzato, le informazioni sulla griglia e la velocità di movimento.
Ha dentro i metodi per muovere il fantasma AI per il labirinto verso il giocatore o farlo scappare dal giocatore a dipendenza di sé è in uno stato vittimizzato, un metodo per ricavare le informazioni della griglia ed una per piazzare il fantasma nella sua posizione iniziale.
4. **PillManager:** Ha dentro i metodi per poter piazzare le pillole e le super pillole sulla griglia.
5. **MenuManager:** Ha dentro tutti i metodi per poter cambiare scena dal menu principale.
6. **SettingsManager:** Contiene tutte le variabili degli slider ed il testo per poter modificare le variabili di gioco ed ha tutti i metodi per poter manipolare e salvare i nuovi valori.
7. **GameModeManager:** Ha dentro i metodi per poter settare la difficoltà di gioco o per andare alla pagina dei settaggi.

4 Implementazione

Nella nostra implementazione abbiamo scritto in dettaglio passo per passo come abbiamo fatto il progetto. Abbiamo voluto quindi incorporare il manuale di costruzione in questa sezione.

4.1 Scripts

Abbiamo riassunto ogni script per ogni metodo in modo dettagliato così da non dovere andare a cercare i dettagli del codice.

4.1.1 Grid Manager

La classe *GridManager* è la classe più importante per il gioco perché gestisce praticamente tutto e poi le altre classi dipendono su *GridManager* per il loro funzionamento.

4.1.1.1 Variabile Costanti

```
public const float PAC_AURA = -0.5f;
public const int INVALID = -1;
public const int WALL = -2;
public const int BLINKY = -3;
public const int PAC_AURA_DIST = 2;
```

Queste costanti sono create per specificare i valori diversi per l'algoritmo [Manhattan Distance](#) ogni volta che attraversa il *Grid*¹.

4.1.1.2 Game Setting Variables

```
[HideInInspector]
public float WallPercent = 0.3f;
[HideInInspector]
public float SuperPillPercent = 0.25f;
[HideInInspector]
public float SuperPillEffectTime = 10f;
[HideInInspector]
public int Lives = 3;
[HideInInspector]
public int LivesGained = 1;
[HideInInspector]
public float BlinkySpeed = 0.5f;
[HideInInspector]
public float PacManSpeed = 0.25f;
```

Queste sono le variabili che il giocatore potrà modificare nel *Game Settings* e che cambiano leggermente la modalità di gioco.

4.1.1.3 Variabili Pubbliche

```
public GameObject LivesPrefab;
public GameObject HighScorePrefab;
public GameObject PointsPrefab;
public Sprite sprite;
public float[,] Grid;
public float[,] InverseGrid;
public bool[,] GridWalls;
[HideInInspector]
public int Points = 0;
```

¹ *Grid* è un array multidimensionale che contiene le posizioni delle mura, pillole, giocatore e fantasma. È la griglia del gioco.

```
[HideInInspector]
public int HighScore;
[HideInInspector]
public int Columns, Rows, GridSize;
[HideInInspector]
public int BlinkyFromPac;
```

Queste sono le variabili che sono accessibili agli altri script.

Le variabili con *HideInInspector* e array multidimensionali non vengono mostrate però nell'interfaccia di *Unity*, mentre le altre, vengono inserite tramite l'interfaccia.

4.1.1.4 Variabili Private

```
private GameObject lives;
private GameObject highScore;
private GameObject points;
private TextMesh livesText;
private TextMesh highScoreText;
private TextMesh pointsText;
```

Queste variabili non sono accessibili agli altri script e il loro valore viene inserite tramite l'interfaccia di *Unity*.

4.1.1.5 Start()

```
void Start()
{
    GetPlayerSettings();

    // Set grid
    Columns = Rows = GridSize;
    Grid = new float[Columns, Rows];
    GridWalls = new bool[Columns, Rows];

    PlaceWalls();

    PlaceGrid();

    PlaceTexts();
}
```

Questo metodo viene avviato all'inizio del gioco:

1. Esegue il metodo [GetPlayerSettings\(\)](#).
2. Imposta la grandezza del *Grid*, ottenuto dal metodo precedente.
3. Esegue i metodi [PlaceWalls\(\)](#), [PlaceGrid\(\)](#) e [PlaceTexts\(\)](#).

4.1.1.6 Update()

```
void Update()
{
    UpdateText();
}
```

Il metodo *Update()* viene eseguito ad ogni *FPS (Frame per Second)* che poi esegue il metodo [UpdateText\(\)](#).

4.1.1.7 GetPlayerSettings()

```
private void GetPlayerSettings()
{
    GridSize = PlayerPrefs.GetInt("GridSize");
    WallPercent = PlayerPrefs.GetFloat("WallPercent");
    BlinkySpeed = PlayerPrefs.GetFloat("BlinkySpeed");
    PacManSpeed = PlayerPrefs.GetFloat("PacManSpeed");
    Lives = PlayerPrefs.GetInt("PlayerLives");
    LivesGained = PlayerPrefs.GetInt("LivesGained");
    SuperPillPercent = PlayerPrefs.GetFloat("SuperPillPercent");
    SuperPillEffectTime = PlayerPrefs.GetInt("SuperPillDuration");
    BlinkyFromPac = GridSize / 2;
}
```

Questo metodo prende i valori da *PlayerPrefs* (i valori che non vengono cancellati alla chiusura dell'applicazione) e li assegna a delle variabili che verranno utilizzati in seguito per cambiare la modalità di gioco.

Alla fine del metodo assegno la distanza da cui il fantasma dovrà essere generato: nel mio caso ho messo metà griglia cioè almeno metà mappa. Il motivo per quale faccio questo è perché altrimenti se il fantasma potesse essere generato ovunque, potrebbe capitare che viene generato alla stessa o molto vicino alla posizione del *Pac-Man* che risulterebbe in una perdita veloce ed ingiusta.

4.1.1.8 PlaceTexts()

```
private void PlaceTexts()
{
    GameObject parent = new GameObject("Texts");

    lives = Instantiate(LivesPrefab, new Vector3(-10, 11.5f), Quaternion.identity);
    lives.name = "Lives";
    lives.transform.parent = parent.transform;
    livesText = lives.GetComponent<TextMesh>();

    highScore = Instantiate(HighScorePrefab, new Vector3(-4, 11.5f), Quaternion.identity);
    highScore.name = "High Score";
    highScore.transform.parent = parent.transform;
    highScoreText = highScore.GetComponent<TextMesh>();

    points = Instantiate(PointsPrefab, new Vector3(5, 11.5f), Quaternion.identity);
    points.name = "Points";
    points.transform.parent = parent.transform;
    pointsText = points.GetComponent<TextMesh>();
}
```

Il metodo *PlaceTexts()*, come dal nome, piazza dei testi, più specificamente, piazza i testi delle vite, punteggio e punteggio migliore all'avvio di una partita. Prima di tutto, crea un *GameObject*, in questo caso come una cartella, e inserisce i *Prefab* dentro questa cartella, assegnandoli anche la posizione e il nome.

4.1.1.9 UpdateText()

```
private void UpdateText()
{
    livesText.text = "LIVES: " + Lives.ToString();
    pointsText.text = "SCORE: " + Points.ToString();
    HighScore = PlayerPrefs.GetInt("HighScore");
    if(Points > HighScore)
    {
        HighScore = Points;
    }
    highScoreText.text = "HIGH SCORE: " + HighScore.ToString();
}
```

Questo metodo aggiorna il testo del punteggio e vite continuamente e se il punteggio migliore viene battuto, anche quello verrà aggiornato.

4.1.1.10 PlaceWalls()

```
public void PlaceWalls()
{
    for (int i = 0; i < Columns; i++)
    {
        for (int j = 0; j < Rows; j++)
        {
            var rnd = Random.Range(0.0f, 1.0f);
            if (rnd < WallPercent)
            {
                GridWalls[i, j] = true;
                SetGrid(0,0);
                for (int k = 0; k < Columns; k++)
                {
                    for (int l = 0; l < Rows; l++)
                    {
                        if (Grid[k, l] == INVALID)
                        {
                            GridWalls[i, j] = false;
                        }
                    }
                }
            }
            else
            {
                GridWalls[i, j] = false;
            }
        }
    }
}
```

Il metodo *PlaceWalls()* piazza le mura sulla *Grid*, però prima di metterle fa un controllo per vedere quanto dovrebbe essere il percentuale delle mura (un valore impostato dall'utente nell'impostazioni del gioco).

4.1.1.11 SetGrid(int, int)

```
public float[,] SetGrid(int x, int y)
{
    Grid = new float[Columns, Rows];

    for (int i = 0; i < Columns; i++)
    {
        for (int j = 0; j < Rows; j++)
        {
            if (GridWalls[i, j])
            {
                Grid[i, j] = WALL;
            }
            else
            {
                Grid[i, j] = INVALID;
            }
        }
    }
    Visit(Grid, x, y, 0);
    return Grid;
}
```

Questo metodo imposta i diversi valori (se è un muro, invalido oppure una via libera) dentro la variabile *Grid*. Questo serve perché altrimenti il fantasma, giocatore e pillole verrebbero piazzate dovunque sulla griglia di gioco.

4.1.1.12 SetInverseGrid()

```
public float[,] SetInverseGrid(int x, int y)
{
    InverseGrid = new float[Columns, Rows];

    for (int i = 0; i < Columns; i++)
    {
        for (int j = 0; j < Rows; j++)
        {
            if (GridWalls[i, j])
            {
                InverseGrid[i, j] = WALL;
            }
            else
            {
                InverseGrid[i, j] = INVALID;
            }
        }
    }

    var max = Grid.Cast<float>().Max();
    for (int i = 0; i < Columns; i++)
    {
        for (int j = 0; j < Rows; j++)
        {
            if (Grid[i, j] == max)
            {
                Visit(InverseGrid, i, j, 0);
            }
        }
    }

    PlaceAura(InverseGrid, x, y);
    return InverseGrid;
}
```

Il metodo *SetInverseGrid()* prendi i dati della *Grid* è, come dal nome, li inverte. Questo lo fa così il fantasma IA si allontana dalla posizione del giocatore, invece di avvicinarsi.

4.1.1.13 PlaceAura()

```
private void PlaceAura(float[,] tempGrid, int row, int col)
{
    for (int i = row - PAC_AURA_DIST; i < row + PAC_AURA_DIST + 1; i++)
    {
        for (int j = col - PAC_AURA_DIST; j < col + PAC_AURA_DIST + 1; j++)
        {
            if (i < Columns && i >= 0 && j < Rows && j >= 0 && tempGrid[i, j] != WALL)
            {
                tempGrid[i, j] = PAC_AURA;
            }
        }
    }
}
```

Questo metodo serve per mettere un'aura attorno al *Pac-Man*, così il fantasma non prova a scappare verso il *Pac-Man*, se il giocatore lo blocca in un angolo.

4.1.1.14 Visit()

```
public void Visit(float[,] tempGrid, int col, int row, int dist)
{
    if (row < Rows && row >= 0 && col < Columns && col >= 0)
    {
        if (
            (tempGrid[col, row] > dist && tempGrid[col, row] != WALL) ||
            tempGrid[col, row] == INVALID
        )
        {
            tempGrid[col, row] = dist;
            Visit(tempGrid, col, row - 1, dist + 1);
            Visit(tempGrid, col - 1, row, dist + 1);
            Visit(tempGrid, col, row + 1, dist + 1);
            Visit(tempGrid, col + 1, row, dist + 1);
        }
    }
}
```

Il metodo *Visit()* è un metodo ricorsivo che usa l'algoritmo *Manhattan Distance*.

4.1.1.14.1 Manhattan Distance

Il *Manhattan Distance* un concetto geometrico, secondo il quale la distanza tra due punti è la somma del valore assoluto delle differenze delle loro coordinate.

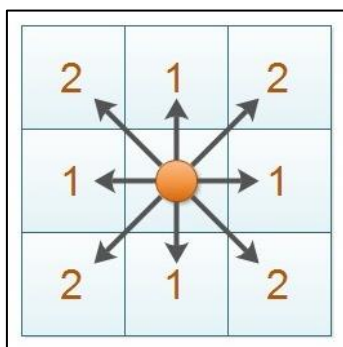


Figura 9 - Manhattan Distance

Il nome è riferito al sistema stradale tipico di luoghi come l'isola di *Manhattan*, in cui gran parte delle vie di scorrimento sono ortogonali tra di loro.

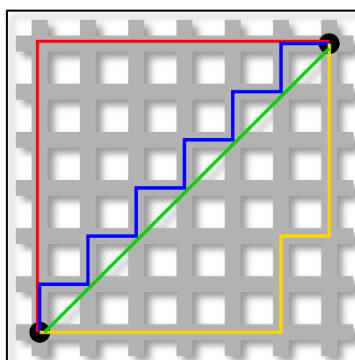


Figura 10 - Manhattan Mapper

4.1.1.15 SpawnTile()

```
private void SpawnTile(int x, int y, Transform parent)
{
    GameObject g = new GameObject("Tile - x: " + x + ",y: " + y);
    g.transform.parent = parent;
    g.transform.position = new Vector3(x - (Columns / 2 - 0.5f), (Rows / 2 - 0.5f) - y);
    var s = g.AddComponent<SpriteRenderer>();
    s.sprite = sprite;
    if (Grid[x, y] == WALL)
    {
        s.color = new Color(0, 0, 255);
    }
    else
    {
        s.color = new Color(0, 0, 0);
    }
}
```

Questo metodo piazza un quadrato della griglia. Li piazza blu se c'è un muro e nero se è una via libera.

4.1.1.16 PlaceGrid()

```
public void PlaceGrid()
{
    // Unpause game
    Time.timeScale = 1f;

    // Game border
    GameObject parent = new GameObject("Grid");
    parent.transform.position = new Vector3(0, 0);
    parent.transform.localScale = new Vector3(Columns + 0.25f, Rows + 0.25f);
    var s = parent.AddComponent<SpriteRenderer>();
    s.sprite = sprite;
    s.sortingOrder = -5;
    if(GridSize == 20)
    {
        s.color = new Color(0, 0, 255);
    }
    else
    {
        s.color = new Color(0, 0, 0);
    }

    // Place Tiles
    for (int i = 0; i < Columns; i++)
    {
        for (int j = 0; j < Rows; j++)
        {
            SpawnTile(i, j, parent.transform);
        }
    }
}
```

Il metodo *PlaceGrid()* richiama il metodo [SpawnTile\(\)](#) per ogni posizione della griglia e aggiunge un bordo attorno alla griglia, così da creare una mappa in cui si può giocare. Viene richiamato ogni volta in cui la mappa deve essere rigenerato.

4.1.2 Pac-Man Manager

Lo Script *PacManManger* serve a gestire tutto ciò che riguarda il personaggio che controlla il giocatore, si occupa di fare tutti i controlli sul dove può e non può andare, consumare le pillole e gestire la vincita e la perdita del giocatore.

4.1.2.1 Variabili Pubbliche

```
public GameObject PacManPrefab;
public GameObject PauseCanvasPrefab;
public GameObject GameOverCanvasPrefab;
public Sprite PacManSprite;
public Sprite PacManSprite2;
public float[,] Grid;
public float[,] InverseGrid;
[HideInInspector]
public int x = 0;
[HideInInspector]
public int y = 0;
```

Ci sono 3 variabili pubbliche di tipo *GameObject* per il *Prefab* dell'interfaccia di gioco, di pausa e di perdita che appare a fine partita. Ci sono 2 variabili di tipo *Sprite* per la *Sprite* di *Pac-Man* con la bocca chiusa ed una per *Pac-Man* con la bocca aperta.

Ci sono 2 matrici di tipo *float* per la griglia (*Grid*) e la griglia inversa (*InverseGrid*) e poi 2 interi per le coordinate di *Pac-Man*.

4.1.2.2 Variabili private

```
private GameObject pacman;
private GameObject Menu;
private GameObject GOMenu;
private Canvas MenuCanvas;
private Canvas GameOverCanvas;
private int col;
private int row;
private int WALL;
private float timeLeft = 0;
private float pacManSpeed;
private int lives;
private int livesGained;
```

Ci sono 3 variabili private di tipo *GameObject* per *Pac-Man*, il menu di pausa ed il game over.

Ci sono 2 *Canvas* per il menu ed il *game over*, 3 *int* per il numero di colonne, righe e mura, 2 *float* per la velocità di *Pac-Man* e 2 *int* per il numero di vite con cui si parte ed il numero di vite che si guadagna quando si vince una partita.

4.1.2.3 Start()

```
void Start()
{
    GetPauseCanvas();
    GetGameOverCanvas();
    GetGridVariables();
    SpawnPacMan();
}
```

Il metodo di *Start()* richiama i metodi per prendere le [canvas](#), le [variabili](#) e per [piazzare il Pac-Man](#).

4.1.2.4 Update()

```
void Update()
{
    // If the ESC key is pressed pause the game
    if (Input.GetKeyDown(KeyCode.Escape) || Input.GetKeyDown("joystick button 9"))
    {
        PauseGame();
    }

    // If X on joystick is pressed then unpause the game
    if (Input.GetKeyDown("joystick button 1"))
    {
        ResumeGame();
    }

    // If O on joystick is pressed then return to menu
    if (Input.GetKeyDown("joystick button 2"))
    {
        ReturnToMenu();
    }

    // Debug that allows me to win
    if (Input.GetKeyDown(KeyCode.L) && Input.GetKeyDown(KeyCode.P))
    {
        GameWin();
    }

    MovePacMan();
}
```

Nel metodo si controlla se viene premuto “esc” sulla tastiera oppure il bottone *Options* sul *controller* per mettere in pausa il gioco.

Se viene premuto “X” dal *controller* il gioco continua e premendo su “O” si torna al menu.

Infine, viene invocato il metodo [MovePacMan\(\)](#) che muove *Pac-Man*.

4.1.2.5 GetGridVariables()

```
public void GetGridVariables()
{
    var g = GetComponent<GridManager>();
    col = g.Columns;
    row = g.Rows;
    Grid = g.Grid;
    InverseGrid = g.InverseGrid;
    pacManSpeed = g.PacManSpeed;
    WALL = GridManager.WALL;
    lives = g.Lives;
    livesGained = g.LivesGained;
}
```

Il metodo prende il componente [GridManager](#) e poi salva dentro le variabili private i valori che gli servono.

4.1.2.6 GetPauseCanvas()

```
private void GetPauseCanvas()
{
    Menu = Instantiate(PauseCanvasPrefab,
        new Vector3(550, 259.5f, 10),
        Quaternion.identity);
    Menu.name = "MenuCanvas";

    GameObject.Find("ResumeButton").GetComponent<Button>().onClick.AddListener(ResumeGame);
    GameObject.Find("MenuButton").GetComponent<Button>().onClick.AddListener(ReturnToMenu);

    MenuCanvas = Menu.GetComponent<Canvas>();
    MenuCanvas.enabled = false;
}
```

Il metodo istanzia la *prefab* della *canvas* di pausa dandogli una posizione ed il nome "MenuCanvas".

Aggiunge i *Listener* ai bottoni per riprendere la partita e per tornare al menu.

Per fare in modo che il *canvas* non appare quando si inizia la partita viene settato il suo parametro "*enabled*" su *false*.

4.1.2.7 GetGameOverCanvas()

```
private void GetGameOverCanvas()
{
    GOMenu = Instantiate(GameOverCanvasPrefab,
        new Vector3(550, 259.5f, 20),
        Quaternion.identity);
    GOMenu.name = "GameOverCanvas";

    GameObject.Find("GOMenuButton").GetComponent<Button>().onClick.AddListener(ReturnToMenu);

    GameOverCanvas = GOMenu.GetComponent<Canvas>();
    GameOverCanvas.enabled = false;
}
```

Funziona allo stesso modo del metodo [GetPauseCanvas\(\)](#). Istanza la *Canvas* prendendo il *prefab*.

Aggiunge i *Listener* ai bottoni e poi mette *false* il parametro "*enabled*" del *canvas* per renderlo non visibile.

4.1.2.8 PauseGame()

```
public void PauseGame()
{
    if(Time.timeScale == 1f)
    {
        Time.timeScale = 0f;
        MenuCanvas.enabled = true;
    }
    else
    {
        Time.timeScale = 1f;
        MenuCanvas.enabled = false;
    }
}
```

Il metodo *PauseGame()* guarda se il *timeScale* del gioco è su 1 o 0. Se è su 1 vuol dire che il gioco sta andando quindi ferma il tempo e fa apparire il *canvas* del menu mentre se è su 0 il gioco è fermo e quindi fa ripartire il tempo e toglie il *canvas* del menu.

4.1.2.9 ResumeGame()

```
public void ResumeGame()
{
    Time.timeScale = 1f;
    MenuCanvas.enabled = false;
}
```

Imposta il *timeScale* su 1 per riprendere il tempo e disabilita la visibilità del *Canvas* del menu.

4.1.2.10 ReturnToMenu()

```
public void ReturnToMenu()
{
    Time.timeScale = 1f;
    SceneManager.LoadScene(0);
}
```

Imposta il *timeScale* su 1 in caso che si sta tornando al menu mentre il tempo è fermo e poi carica la Scena 0 usando lo [SceneManager](#).

4.1.2.11 GameOver()

```
public void GameOver()
{
    Time.timeScale = 0f;
    GameObject.Find("GOScore").GetComponent<TextMeshProUGUI>().text = "Score: " +
    GetComponent<GridManager>().Points.ToString();
    GameObject.Find("GOHighScore").GetComponent<TextMeshProUGUI>().text = "HighScore: " +
    GetComponent<GridManager>().HighScore.ToString();
    PlayerPrefs.SetInt("HighScore", GetComponent<GridManager>().HighScore);
    GameOverCanvas.enabled = true;
}
```

Blocca il tempo e poi aggiorna il proprio punteggio ed il punteggio massimo sulla schermata di *GameOver* prima di riabilitarla.

4.1.2.12 GameWin()

```
public void GameWin()
{
    Destroy(GameObject.Find("Grid"));
    GetComponent<GridManager>().PlaceWalls();
    GetComponent<GridManager>().PlaceGrid();

    Destroy(GameObject.Find("Pills"));
    GetComponent<PillManager>().PlacePills();

    x = 0;
    y = 0;
    GetComponent<EnemyManager>().ResetBlinkyPosition();
    lives += livesGained;
    GetComponent<GridManager>().Lives = lives;
}
```

Distrukge la griglia e poi piazza una nuova griglia con delle nuove mura. Distrukge le pillole e piazza nuove pillole, setta le coordinate di *Pac-Man* a 0 e poi resetta la posizione del fantasma. Aggiunge al numero di vite il numero di vite che si guadagna alla vincita di una partita e poi setta questo numero anche alle avariabili nel [GridManager](#).

4.1.2.13 MovePacMan()

```
private void MovePacMan()
{
    // Do all checks
    CheckPacManEatsPill(x - (col / 2 - 0.5f), (row / 2 - 0.5f) - y);
    CheckPacManEatsSuperPill(x - (col / 2 - 0.5f), (row / 2 - 0.5f) - y);
    CheckPacManEncountersBlinky(x - (col / 2 - 0.5f), (row / 2 - 0.5f) - y);

    pacman.GetComponent<SpriteRenderer>().sprite = PacManSprite2;

    // Makes timer go down and when the wait time is over Lets Pac-Man move again
    timeLeft -= Time.deltaTime;
    if (timeLeft < 0)
    {
        // Check what key is pressed and checks if the movement makes Pac-Man go out of the borders
        if (Input.GetAxis("Vertical") < 0 && y < row - 1)
        {
            if (CheckIfNotWall(x, y + 1))
            {
                y++;
                pacman.transform.eulerAngles = Vector3.forward * -90;
                pacman.GetComponent<SpriteRenderer>().sprite = PacManSprite2;
            }
        }
        else if (Input.GetAxis("Horizontal") > 0 && x < col - 1)
        {
            if (CheckIfNotWall(x + 1, y))
            {
                x++;
                pacman.transform.eulerAngles = Vector3.forward;
                pacman.GetComponent<SpriteRenderer>().sprite = PacManSprite2;
            }
        }
        else if (Input.GetAxis("Vertical") > 0 && y > 0)
        {
            if (CheckIfNotWall(x, y - 1))
            {
                y--;
                pacman.transform.eulerAngles = Vector3.forward * 90;
                pacman.GetComponent<SpriteRenderer>().sprite = PacManSprite2;
            }
        }
        else if (Input.GetAxis("Horizontal") < 0 && x > 0)
        {
            if (CheckIfNotWall(x - 1, y))
            {
                x--;
                pacman.transform.eulerAngles = Vector3.forward * 180;
                pacman.GetComponent<SpriteRenderer>().sprite = PacManSprite2;
            }
        }
    }

    // Reset wait time
    timeLeft = pacManSpeed;

    // Set Pac-Man position
    pacman.transform.position = new Vector3(
        x - (col / 2 - 0.5f),
        (row / 2 - 0.5f) - y);

    // Update the Grid (for the AI)
    Grid = GetComponent<GridManager>().SetGrid(x, y);
    InverseGrid = GetComponent<GridManager>().SetInverseGrid(x, y);
}

// Once half of the wait time is over the sprite is changed
if (timeLeft < pacManSpeed / 2)
{
    pacman.GetComponent<SpriteRenderer>().sprite = PacManSprite;
}
```

```
// If all pills are eaten
var pills = GameObject.FindGameObjectsWithTag("Pill");
var sPills = GameObject.FindGameObjectsWithTag("SuperPill");
if(pills.Length == 0 && sPills.Length == 0)
{
    GameWin();
}
```

All'inizio del metodo invoca 3 metodi per fare dei controlli sulla cella su cui si trova e cambia la *Sprite* di *Pac-Man*. Sottrae il tempo trascorso ad una variabile alla quale viene settato un tempo di attesa ogni volta che il giocatore si muove. Quando questa variabile va sotto 0 e quindi il tempo è trascorso il giocatore può muoversi. Successivamente viene eseguito un controllo per vedere a che direzione i tasti premuti corrispondono e per vedere che il movimento in quella direzione non porti fuori dalla griglia. Se il controllo passa viene controllato se la prossima cella è un muro con il metodo [CheckIfNotWall\(\)](#) e se passa anche questo controllo viene modificata la posizione e cambiata la direzione in cui guarda *Pac-Man*. Dopo i controlli viene cambiata la *Sprite* ogni tot tempo e viene preso il numero di pillole e super pillole per vedere se sono state consumate tutte e quindi invocare il metodo [GameWin\(\)](#).

4.1.2.14 CheckIfNotWall()

```
public bool CheckIfNotWall(int x, int y)
{
    return (Grid[x, y] != WALL);
}
```

Questo metodo prende come parametro le coordinate di una cella e controlla dentro la matrice della griglia se la cella è un muro.

4.1.2.15 SpawnPacMan()

```
public void SpawnPacMan()
{
    pacman = Instantiate(PacManPrefab, new Vector3(
        x - (col / 2 - 0.5f),
        (row / 2 - 0.5f) - y), Quaternion.identity);
    pacman.name = "Pac-Man";
    timeLeft = 1;
}
```

Il metodo istanzia *Pac-Man* in cima a sinistra della mappa, dà il nome "*Pac-Man*" all'oggetto e setta il tempo di attesa per il movimento ad 1.

4.1.2.16 CheckPacManEatsPill()

```
public void CheckPacManEatsPill(float pacX, float pacY)
{
    var pills = GameObject.FindGameObjectsWithTag("Pill");
    foreach(var obj in pills)
    {
        float pillX = obj.transform.position.x;
        float pillY = obj.transform.position.y;
        if(pacX == pillX && pacY == pillY)
        {
            GetComponent<GridManager>().Points++;
            Destroy(obj);
        }
    }
}
```

Questo metodo serve a controllare se *Pac-Man* è su una cella con una pillola. Prende tutti gli oggetti con il tag "*Pill*" e gli scorre in un ciclo. Se le coordinate della pillola corrispondono alle coordinate di *Pac-Man* l'oggetto viene distrutto ed il punteggio aumenta.

4.1.2.17 CheckPacManEatsSuperPill()

```
public void CheckPacManEatsSuperPill(float pacX, float pacY)
{
    var pills = GameObject.FindGameObjectsWithTag("SuperPill");
    foreach (var obj in pills)
    {
        float pillX = obj.transform.position.x;
        float pillY = obj.transform.position.y;
        if (pacX == pillX && pacY == pillY)
        {
            GetComponent<GridManager>().Points++;
            Destroy(obj);
            var enemy = GetComponent<EnemyManager>();
            if(!enemy.isScared && !enemy.isEaten)
            {
                enemy.isScared = true;
            }
            GetComponent<EnemyManager>().timeLeft = GetComponent<GridManager>().SuperPillEffectTime;
        }
    }
}
```

Questo metodo serve a controllare se *Pac-Man* è su una cella con una super pillola. Prende tutti gli oggetti con il tag "*SuperPill*" e gli scorre in un ciclo. Se le coordinate della super pillola corrispondono alle coordinate di *Pac-Man* l'oggetto viene distrutto, il punteggio aumenta e viene fatto un controllo sul se il fantasma è mangiato o se è già in stato vittimizzato, altrimenti viene settata la variabile a *true*. Il tempo di vittimizzazione viene settato dentro [l'EnemyManager](#).

4.1.2.18 CheckPacManEncountersBlinky()

```
public void CheckPacManEncountersBlinky(float pacX, float pacY)
{
    var blinky = GameObject.FindGameObjectWithTag("Blinky");
    float blinkyX = blinky.transform.position.x;
    float blinkyY = blinky.transform.position.y;
    if(pacX == blinkyX && pacY == blinkyY)
    {
        // If Blinky is scared
        if (GetComponent<EnemyManager>().isScared)
        {
            GetComponent<GridManager>().Points += 5;
            GetComponent<EnemyManager>().isEaten = true;
            GetComponent<EnemyManager>().timeLeft = 5;
        }
        else
        {
            // If Pac-Man still has live reset position otherwise Game Over
            if (lives > 0)
            {
                lives--;
                x = 0;
                y = 0;
                pacman.transform.position = new Vector3(
                    x - (col / 2 - 0.5f),
                    (row / 2 - 0.5f) - y);
                GetComponent<EnemyManager>().ResetBlinkyPosition();
                GetComponent<GridManager>().Lives = lives;

                StartCoroutine(WaitASec());
            }
            else
            {
                GameOver();
            }
        }
    }
}
```

Viene salvato il fantasma in una variabile dalla quale vengono estrapolate le coordinate in altre due variabili. Poi fa un controllo per vedere se le coordinate di *Pac-Man* sono uguali a quelle del fantasma e se si viene controllato se il fantasma è in stato vittimizzato. Se lo è allora aumenta il punteggio di 5, viene settata a *true* una variabile per mostrare che il fantasma è stato mangiato e settato a 5 il tempo per cui rimane in questo stato. Altrimenti viene controllato se il giocatore ha delle vite, se si gli viene tolta una vita, viene resettata la sua posizione e la posizione del fantasma e viene fatta partire una *Coroutine* con il metodo [WaitASec\(\)](#).

4.1.2.19 WaitASec()

```
private IEnumerator WaitASec()
{
    Time.timeScale = 0f;
    yield return new WaitForSecondsRealtime(1);
    Time.timeScale = 1f;
}
```

Viene bloccato il tempo e poi il metodo aspetta un secondo prima di far riprendere il tempo.

4.1.3 Enemy Manager

Questo *script* gestisce tutto quello inerente al IA fantasma (il nemico chiamato *Blinky*²).

4.1.3.1 Variabili Pubbliche

```
public GameObject blinkyPrefab;
public Sprite sprite;
public RuntimeAnimatorController BlinkyController;
[HideInInspector]
public bool isScared = false;
[HideInInspector]
public bool isEaten = false;
[HideInInspector]
public float timeLeft = 0;
```

Queste variabili sono pubbliche e perciò sono viste dagli altri *script*.

Per *EnemyManager* gli passo il *prefab*, *Sprite* e *Animator Controller* attraverso l'interfaccia di *Unity*. Le altre variabili servono per dire in che stato è *Blinky* (spaventato o mangiato) e per quanto tempo rimarrà in quel stato.

4.1.3.2 Variabili Private

```
private GameObject blinky;
private Animator blinkyAnimator;
private int blinkyX;
private int blinkyY;
private float timer;
private bool[,] gridWalls;
private float[,] grid;
private int blinkyFromPac;
private int columns;
private int rows;
private int WALL;
private float PAC_AURA;
private float blinkySpeed;
private int runawaySpeed;
private float superPillEffectTime;
private bool isFirstTime = true;
```

Queste variabili sono nascoste dagli altri *script* e perciò servono solo per *EnemyManager*. Gran parte delle variabili sono prese usando il metodo [GetGridVariables\(\)](#) mentre delle altre importanti sono *blinkyX* e *blinkyY* che specificano la posizione di *Blinky*.

4.1.3.3 Start()

```
void Start()
{
    GetGridVariables();
    ResetBlinkyPosition();
    SpawnBlinky(blinkyX, blinkyY);
    isFirstTime = false;
    timeLeft = superPillEffectTime;
}
```

Come detto precedente, il metodo *Start()* viene richiamato solo una volta all'avvio dello *script* e fa le seguenti cose:

1. Esegue i metodi [GetGridVariables\(\)](#), [ResetBlinkyPosition\(\)](#) e [SpawnBlinky\(\)](#).
2. Imposta la durata dell'effetto della super pillola (che puoi specificare nel *Game Mode* dentro il gioco).

² *Blinky* era il nome dato al fantasma rosso nel *Pac-Man* originale.

4.1.3.4 Update()

```
void Update()
{
    MoveBlinky();
}
```

Questo metodo viene richiamato ad ogni *FPS* ed esegue il metodo [MoveBlinky\(\)](#).

4.1.3.5 GetGridVariables()

```
public void GetGridVariables()
{
    var g = GetComponent<GridManager>();
    gridWalls = g.GridWalls;
    grid = g.Grid;
    blinkyFromPac = g.BlinkyFromPac;
    columns = g.Columns;
    rows = g.Rows;
    WALL = GridManager.WALL;
    PAC_AURA = GridManager.PAC_AURA;
    blinkySpeed = g.BlinkySpeed;
    superPillEffectTime = g.SuperPillEffectTime;
}
```

Prende i valori dal componente [GridManager](#) e li assegna alle variabili istanziate prima come private.

4.1.3.6 ResetBlinkyPosition()

```
public void ResetBlinkyPosition()
{
    // Goes through all the Positions (from Preset Distance) to find one without a Wall
    do
    {
        blinkyX = UnityEngine.Random.Range(blinkyFromPac, columns - 1);
        blinkyY = UnityEngine.Random.Range(blinkyFromPac, rows - 1);
    } while (gridWalls[blinkyX, blinkyY]);

    if (!isFirstTime)
    {
        SetBlinkyPosition(blinkyX, blinkyY);
    }
    isScared = false;
    isEaten = false;
}
```

Il metodo *ResetBlinkyPosition()* attraversa tutte le posizioni con una via libera al di fuori dalla distanza del giocatore (nel mio caso 10) e perciò *Blinky* potrà solo essere piazzato 10+ posizioni dal *Pac-Man*.

4.1.3.7 SpawnBlinky()

```
private void SpawnBlinky(int x, int y)
{
    blinky = Instantiate(blinkyPrefab, new Vector3(
        x - (columns / 2 - 0.5f),
        (rows / 2 - 0.5f) - y), Quaternion.identity);
    blinky.name = "Blinky";
    blinkyAnimator = blinky.GetComponent<Animator>();
}
```

Questo metodo crea il *GameObject Blinky* e lo piazza sulla griglia.

4.1.3.8 SetBlinkyPosition()

```
private void SetBlinkyPosition(int x, int y)
{
    blinky.transform.position = new Vector3(
        x - (columns / 2 - 0.5f),
        (rows / 2 - 0.5f) - y
    );
}
```

Il metodo *SetBlinkyPosition()* imposta la posizione del fantasma attraverso due coordinate come parametri.

4.1.3.9 MoveBlinky()

```
private void MoveBlinky()
{
    // Timer for how long the ghost should be scared for
    if (isScared)
    {
        timeLeft -= Time.deltaTime;
        if (timeLeft <= 0)
        {
            isScared = false;
            isEaten = false;
            timeLeft = superPillEffectTime;
        }
    }

    // Get updated Grid
    grid = GetComponent<PacManManager>().Grid;
    if (isScared || isEaten)
    {
        grid = GetComponent<PacManManager>().InverseGrid;
    }

    // If x time hasn't passed don't execute the method
    runawaySpeed = 1;
    if (isEaten)
    {
        runawaySpeed = 4;
    }
    if (timer < blinkySpeed / runawaySpeed)
    {
        timer += Time.deltaTime;
        return;
    }
    else
    {
        timer = 0;
    }
}
```

Questo parte del metodo controlla per quanto tempo il fantasma dovrebbe rimanere spaventato. Poi prende la Grid dal componente [GridManager](#) con la posizione che *Blinky* dovrà proseguire. E poi vede se il fantasma è stato mangiato e gli quadruplica la velocità.

```
float north = WALL;
float west = WALL;
float south = WALL;
float east = WALL;

// Assign if the positions are in the field
if (blinkyY > 0)
{
    north = grid[blinkyX, blinkyY - 1];
}
if (blinkyX > 0)
{
    west = grid[blinkyX - 1, blinkyY];
}
```

```

    }
    if (blinkyY < rows - 1)
    {
        south = grid[blinkyX, blinkyY + 1];
    }
    if (blinkyX < columns - 1)
    {
        east = grid[blinkyX + 1, blinkyY];
    }

    // Positions which are walls or invalid
    float avoid = columns * 1000;
    if (north == WALL || north == PAC_AURA)
    {
        north = Math.Abs(north * avoid);
    }
    if (west == WALL || west == PAC_AURA)
    {
        west = Math.Abs(west * avoid);
    }
    if (south == WALL || south == PAC_AURA)
    {
        south = Math.Abs(south * avoid);
    }
    if (east == WALL || east == PAC_AURA)
    {
        east = Math.Abs(east * avoid);
    }
}

```

In questa parte del codice viene controllato che il fantasma non esce dalla mappa e che non attraversa le mura ma invece segue il percorso come un labirinto.

```

blinkyAnimator.SetBool("isUp", false);
blinkyAnimator.SetBool("isRight", false);
blinkyAnimator.SetBool("isLeft", false);
blinkyAnimator.SetBool("isDown", false);

blinkyAnimator.SetBool("isEyesUp", false);
blinkyAnimator.SetBool("isEyesRight", false);
blinkyAnimator.SetBool("isEyesLeft", false);
blinkyAnimator.SetBool("isEyesDown", false);

blinkyAnimator.SetBool("isScared", false);

// Blinky going up
if (north <= west && north <= east && north <= south)
{
    if (isScared && !isEaten)
    {
        blinkyAnimator.SetBool("isScared", true);
    }
    else
    {
        if (isEaten)
        {
            blinkyAnimator.SetBool("isEyesUp", true);
        }
        else
        {
            blinkyAnimator.SetBool("isUp", true);
        }
    }

    blinkyY--;
}
// Blinky going left
else if (west <= north && west <= east && west <= south)
{
    if (isScared && !isEaten)
    {

```

```

        blinkyAnimator.SetBool("isScared", true);
    }
    else
    {
        if (isEaten)
        {
            blinkyAnimator.SetBool("isEyesLeft", true);
        }
        else
        {
            blinkyAnimator.SetBool("isLeft", true);
        }
    }
    blinkyX--;
}
// Blinky going right
else if (east <= west && east <= north && east <= south)
{
    if (isScared && !isEaten)
    {
        blinkyAnimator.SetBool("isScared", true);
    }
    else
    {
        if (isEaten)
        {
            blinkyAnimator.SetBool("isEyesRight", true);
        }
        else
        {
            blinkyAnimator.SetBool("isRight", true);
        }
    }
    blinkyX++;
}
// Blinky going down
else
{
    if (isScared && !isEaten)
    {
        blinkyAnimator.SetBool("isScared", true);
    }
    else
    {
        if (isEaten)
        {
            blinkyAnimator.SetBool("isEyesDown", true);
        }
        else
        {
            blinkyAnimator.SetBool("isDown", true);
        }
    }
    blinkyY++;
}
}

```

Poi fa dei controlli per vedere in che stato preciso è *Blinky* e gli assegna la sua rispettiva animazione (vedi [Animazioni](#) per ulteriori dettagli).

```

    blinky.transform.position = new Vector3(
        blinkyX - (columns / 2 - 0.5f),
        (rows / 2 - 0.5f) - blinkyY
    );
}

```

Infine assegna la posizione del fantasma.
Tutto questo metodo viene ripetuto and ogni *FPS*.

4.1.4 Pill Manager

Questa classe gestisce le pillole e le super pillole, piazzandole sulla griglia dopo che viene generata.

4.1.4.1 Variabili Pubbliche

```
public Sprite PillSprite;
public GameObject PillPrefab;
public GameObject SuperPillPrefab;
public float PillSize = 0.15f;
public float SuperPillSize = 0.5f;
```

Queste sono le variabili che sono accessibili agli altri script.

Tutte variabili vengono inserite tramite l'interfaccia. Gli ultimi due variabili specificano le grandezze delle pillole.

4.1.4.2 Variabili Private

```
private bool[,] gridWalls;
private int columns;
private int rows;
private float superPillPercent;
```

Queste variabili sono nascoste dagli altri script. Sono prese usando il metodo [GetGridVariables\(\)](#)

4.1.4.3 Start()

```
void Start()
{
    GetGridVariables();
    PlacePills();
}
```

All'avvio dell'applicazione vengono eseguite i metodi [GetGridVariables\(\)](#) e [PlacePills\(\)](#)

4.1.4.4 GetGridVariables()

```
public void GetGridVariables()
{
    var g = GetComponent<GridManager>();
    gridWalls = g.GridWalls;
    columns = g.Columns;
    rows = g.Rows;
    superPillPercent = g.SuperPillPercent;
}
```

Il metodo prende il componente [GridManager](#) e poi salva dentro le variabili private i valori che gli servono.

4.1.4.5 SpawnPill()

```
private void SpawnPill(int x, int y, bool isSuper, Transform parent)
{
    GameObject g;
    // Spawn Super Pill
    if (isSuper)
    {
        g = Instantiate(SuperPillPrefab, new Vector3(
            x - (columns / 2 - 0.5f),
            (rows / 2 - 0.5f) - y), Quaternion.identity);
        g.name = "Super Pill - x: " + x + ", y: " + y;
    }
    // Spawn Normal Pill
    else
    {

```

```
g = Instantiate(PillPrefab, new Vector3(
    x - (columns / 2 - 0.5f),
    (rows / 2 - 0.5f) - y), Quaternion.identity);
g.name = "Pill - x: " + x + ",y: " + y;
}
g.transform.parent = parent;
}
```

Piazza una pillola nella griglia e viene specificato tramite i parametri: la posizione, il *parent* (contenitore per tenerle organizzate) e se è una super pillola o no. A dipendenza di questo, cambierà il nome e la grandezza.

4.1.4.6 PlacePills()

```
public void PlacePills()
{
    GameObject parent = new GameObject("Pills");
    for (int i = 0; i < columns; i++)
    {
        for (int j = 0; j < rows; j++)
        {
            //if there isn't a Wall
            if (GetComponent<GridManager>().Grid[i, j] != GridManager.WALL)
            {
                // if it's corned place a Super Pill sometimes
                if (
                    !(i == 0 && j == 0) && (
                        (j - 1 < 0 || gridWalls[i, j - 1] == true) &&
                        (i - 1 < 0 || gridWalls[i - 1, j] == true) &&
                        (j + 1 > rows - 1 || gridWalls[i, j + 1] == true) ||
                        (j - 1 < 0 || gridWalls[i, j - 1] == true) &&
                        (i + 1 > columns - 1 || gridWalls[i + 1, j] == true) &&
                        (j + 1 > rows - 1 || gridWalls[i, j + 1] == true) ||
                        (j - 1 < 0 || gridWalls[i, j - 1] == true) &&
                        (i - 1 < 0 || gridWalls[i - 1, j] == true) &&
                        (i + 1 > columns - 1 || gridWalls[i + 1, j] == true) ||
                        (j + 1 > rows - 1 || gridWalls[i, j + 1] == true) &&
                        (i - 1 < 0 || gridWalls[i - 1, j] == true) &&
                        (i + 1 > rows - 1 || gridWalls[i + 1, j] == true)
                    )
                )
                {
                    var rnd = Random.Range(0.0f, 1.0f);
                    if (rnd < superPillPercent)
                    {
                        SpawnPill(i, j, true, parent.transform);
                    }
                    else
                    {
                        SpawnPill(i, j, false, parent.transform);
                    }
                }
            }
            else
            {
                SpawnPill(i, j, false, parent.transform);
            }
        }
    }
}
```

Questo metodo richiama il metodo precedente per ogni posizione nella griglia che è una via libera. Poi piazza le super pillole ad ogni vicolo cieco, però dipende della percentuale assegnato nel menu *Game Mode*.

4.1.5 Game Mode Manager

Lo script *GameModeManager* si occupa di gestire le diverse difficoltà di gioco per rendere il gioco più accessibili agli utenti.

4.1.5.1 Start()

```
void Start()
{
    GameObject.Find("Easy").GetComponent<Button>().onClick.AddListener(EasyMode);
    GameObject.Find("Medium").GetComponent<Button>().onClick.AddListener(MediumMode);
    GameObject.Find("Hard").GetComponent<Button>().onClick.AddListener(HardMode);
    GameObject.Find("Custom").GetComponent<Button>().onClick.AddListener(Settings);

    GameObject.Find("Custom").SetActive(true);
}
```

Nel metodo *Start* viene assegnato un *Listener* a tutti i bottoni per fare in modo che quando un bottone viene premuto viene eseguito il metodo richiesto.

4.1.5.2 Update()

```
private void Update()
{
    // If [ ] on joystick is pressed then change game mode to easy
    if (Input.GetKeyDown("joystick button 0"))
    {
        EasyMode();
    }

    // If X on joystick is pressed then change game mode to medium
    if (Input.GetKeyDown("joystick button 1"))
    {
        MediumMode();
    }

    // If O on joystick is pressed then change game mode to hard
    if (Input.GetKeyDown("joystick button 2"))
    {
        HardMode();
    }

    // If a joystick is connected the Custom button gets deactivated
    if (MenuManager.isJoystick)
    {
        GameObject.Find("Custom").SetActive(false);
        MenuManager.isJoystick = false;
    }
}
```

Il metodo *Update()* guarda se vengono premuti i pulsanti sul *controller*. Se viene premuto "X" inizia il gioco, se viene premuto "O" esce dall'applicazione e se viene premuto "[]" va alla pagina per cambiare modalità di gioco e viene impostata *true* la variabile *isJoystick*.

4.1.5.3 EasyMode()

```
public void EasyMode()
{
    PlayerPrefs.SetInt("GridSize", 15);
    PlayerPrefs.SetFloat("WallPercent", 0.1f);
    PlayerPrefs.SetFloat("BlinkySpeed", 0.9f);
    PlayerPrefs.SetFloat("PacManSpeed", 0.25f);
    PlayerPrefs.SetInt("PlayerLives", 5);
    PlayerPrefs.SetInt("LivesGained", 5);
    PlayerPrefs.SetFloat("SuperPillPercent", 1f);
    PlayerPrefs.SetInt("SuperPillDuration", 15);
    SceneManager.LoadScene(0);
}
```

Nel metodo *EasyMode()* vengono impostate le *PlayerPrefs* ad i valori per rendere il gioco più facile e poi ti ritorna al menu.

4.1.5.4 MediumMode()

```
public void MediumMode()
{
    PlayerPrefs.SetInt("GridSize", 20);
    PlayerPrefs.SetFloat("WallPercent", 0.3f);
    PlayerPrefs.SetFloat("BlinkySpeed", 0.5f);
    PlayerPrefs.SetFloat("PacManSpeed", 0.25f);
    PlayerPrefs.SetInt("PlayerLives", 3);
    PlayerPrefs.SetInt("LivesGained", 1);
    PlayerPrefs.SetFloat("SuperPillPercent", 0.25f);
    PlayerPrefs.SetInt("SuperPillDuration", 10);
    SceneManager.LoadScene(0);
}
```

Nel metodo *MediumMode()* vengono impostate le *PlayerPrefs* ad i valori per rendere il gioco di difficoltà normale e poi ti ritorna al menu.

4.1.5.5 HardMode()

```
public void HardMode()
{
    PlayerPrefs.SetInt("GridSize", 20);
    PlayerPrefs.SetFloat("WallPercent", 0.4f);
    PlayerPrefs.SetFloat("BlinkySpeed", 0.2f);
    PlayerPrefs.SetFloat("PacManSpeed", 0.1f);
    PlayerPrefs.SetInt("PlayerLives", 1);
    PlayerPrefs.SetInt("LivesGained", 1);
    PlayerPrefs.SetFloat("SuperPillPercent", 0.5f);
    PlayerPrefs.SetInt("SuperPillDuration", 5);
    SceneManager.LoadScene(0);
}
```

Nel metodo *HardMode()* vengono impostate le *PlayerPrefs* ad i valori per rendere il gioco più difficile e poi ti ritorna al menu.

4.1.5.6 Settings()

```
public void Settings()
{
    SceneManager.LoadScene(2);
}
```

Carica la scena per poter modificare le *PlayerPrefs* manualmente con degli *Slider*.

4.1.6 Settings Manager

Lo script *SettingsManager* si occupa di gestire la scena dei settaggi.

4.1.6.1 Variabili

```
// Public variables
public GameObject SettingsCanvasPrefab;

// Private variables
private Slider GS;
private Slider WP;
private Slider BS;
private Slider PS;
private Slider PL;
private Slider LG;
private Slider SPP;
private Slider SPD;
private TextMeshProUGUI GStext;
private TextMeshProUGUI WPtext;
private TextMeshProUGUI BStext;
private TextMeshProUGUI PStext;
private TextMeshProUGUI PLtext;
private TextMeshProUGUI LGtext;
private TextMeshProUGUI SPPtext;
private TextMeshProUGUI SPDtext;
```

L'unica variabile pubblica è di tipo *GameObject* per farsi passare la *prefab* dell'interfaccia.

Poi ci sono 8 variabili private per il valore degli 8 *Slider*, e ci sono altre 8 variabili di tipo *TextMeshProUGUI*³ per poter mostrare il valore degli slider su testo di fianco agli slider.

4.1.6.2 Start()

```
void Start()
{
    GetComponents();
}
```

Il metodo *Start()* richiama il metodo [GetComponents\(\)](#).

³ *TextMeshProGUI* è un'estensione di *Unity* per aggiungere testi avanzate.

4.1.6.3 Update()

```
void Update()
{
    GStext.text = GS.value.ToString();
    WPtext.text = WP.value.ToString() + "%";
    BStext.text = BS.value.ToString();
    PStext.text = PS.value.ToString();
    PLtext.text = PL.value.ToString();
    LGtext.text = LG.value.ToString();
    SPPtext.text = SPP.value.ToString() + "%";
    SPDtext.text = SPD.value.ToString();
}
```

Nel metodo *Update()* viene aggiornato il testo delle *TextMesh* prendendo il valore numerico degli *Slider* e convertendoli in stringa e nel caso delle percentuali viene aggiunto davanti il %.

4.1.6.4 GetComponents()

```
public void GetComponents() {
    var canvas = Instantiate(SettingsCanvasPrefab,
        new Vector3(550, 259.5f, 10),
        Quaternion.identity);
    canvas.name = "Canvas";

    GameObject.Find("ButtonBack").GetComponent<Button>().onClick.AddListener(ReturnToMenu);
    GameObject.Find("ButtonConfirm").GetComponent<Button>().onClick.AddListener(ConfirmEdit);
    GameObject.Find("ButtonDefault").GetComponent<Button>().onClick.AddListener(ResetValues);

    GS = GameObject.Find("SliderGS").GetComponent<Slider>();
    WP = GameObject.Find("SliderWP").GetComponent<Slider>();
    BS = GameObject.Find("SliderBS").GetComponent<Slider>();
    PS = GameObject.Find("SliderPS").GetComponent<Slider>();
    PL = GameObject.Find("SliderPL").GetComponent<Slider>();
    LG = GameObject.Find("SliderLG").GetComponent<Slider>();
    SPP = GameObject.Find("SliderSPP").GetComponent<Slider>();
    SPD = GameObject.Find("SliderSPD").GetComponent<Slider>();

    GStext = GameObject.Find("ValGS").GetComponent<TextMeshProUGUI>();
    WPtext = GameObject.Find("ValWP").GetComponent<TextMeshProUGUI>();
    BStext = GameObject.Find("ValBS").GetComponent<TextMeshProUGUI>();
    PStext = GameObject.Find("ValPS").GetComponent<TextMeshProUGUI>();
    PLtext = GameObject.Find("ValPL").GetComponent<TextMeshProUGUI>();
    LGtext = GameObject.Find("ValLG").GetComponent<TextMeshProUGUI>();
    SPPtext = GameObject.Find("ValSPP").GetComponent<TextMeshProUGUI>();
    SPDtext = GameObject.Find("ValSPD").GetComponent<TextMeshProUGUI>();

    GS.wholeNumbers = true;
    GS.minValue = 15;
    GS.maxValue = 21;

    WP.wholeNumbers = true;
    WP.minValue = 0;
    WP.maxValue = 50;

    BS.wholeNumbers = true;
    BS.minValue = 1;
    BS.maxValue = 6;

    PS.wholeNumbers = true;
    PS.minValue = 1;
    PS.maxValue = 5;

    PL.wholeNumbers = true;
    PL.minValue = 0;
    PL.maxValue = 10;
}
```

```

    LG.wholeNumbers = true;
    LG.minValue = 0;
    LG.maxValue = 5;

    SPP.wholeNumbers = true;
    SPP.minValue = 0;
    SPP.maxValue = 100;

    SPD.wholeNumbers = true;
    SPD.minValue = 5;
    SPD.maxValue = 15;

    GS.value = PlayerPrefs.GetInt("GridSize");
    WP.value = PlayerPrefs.GetFloat("WallPercent") * 100;
    BS.value = (1 - PlayerPrefs.GetFloat("BlinkySpeed")) / 0.125f;
    PS.value = (1 - PlayerPrefs.GetFloat("PacManSpeed")) / 0.0625f - 9;
    PL.value = PlayerPrefs.GetInt("PlayerLives");
    LG.value = PlayerPrefs.GetInt("LivesGained");
    SPP.value = PlayerPrefs.GetFloat("SuperPillPercent") * 100;
    SPD.value = PlayerPrefs.GetInt("SuperPillDuration");
}

```

Nel metodo *GetComponents()* viene istanziata la *prefab* in una variabile con la sua posizione e poi gli viene dato il nome “*Canvas*”.

Viene assegnato un *Listener* a tutti i bottoni per fare in modo che quando un bottone viene premuto viene eseguito il metodo richiesto.

In tutte le variabili di tipo *Slider* viene messo dentro il componente di tipo *Slider*.

In tutte le variabili di testo viene messo dentro il componente di tipo *TextMeshProGUI*.

A tutti gli *Slider* viene messo su *true* il parametro per rendere selezionabili solo numeri interi sullo *Slider*.

Poi viene impostato il valore minimo e massimo per tutti gli *Slider*.

Come valore degli *Slider* al caricamento della pagina vengono presi i valori precedentemente impostati.

4.1.6.5 ReturnToMenu()

```

public void ReturnToMenu()
{
    SceneManager.LoadScene(0);
}

```

Il metodo *ReturnToMenu()* carica la scena del menu.

4.1.6.6 ConfirmEdit()

```

public void ConfirmEdit()
{
    PlayerPrefs.SetInt("GridSize", Mathf.RoundToInt(GS.value));
    PlayerPrefs.SetFloat("WallPercent", WP.value / 100);
    PlayerPrefs.SetFloat("BlinkySpeed", 1 - BS.value * 0.125f);
    PlayerPrefs.SetFloat("PacManSpeed", 1 - (PS.value + 9) * 0.0625f);
    PlayerPrefs.SetInt("PlayerLives", Mathf.RoundToInt(PL.value));
    PlayerPrefs.SetInt("LivesGained", Mathf.RoundToInt(LG.value));
    PlayerPrefs.SetFloat("SuperPillPercent", SPP.value / 100);
    PlayerPrefs.SetInt("SuperPillDuration", Mathf.RoundToInt(SPD.value));
    ReturnToMenu();
}

```

Il metodo *ConfirmEdit()* setta le *PlayerPrefs* come sono state impostate dall'utente usando gli *Slider* e poi ti ritorna al menu.

4.1.6.7 ResetValues()

```
public void ResetValues()
{
    PlayerPrefs.SetInt("GridSize", 20);
    PlayerPrefs.SetFloat("WallPercent", 0.3f);
    PlayerPrefs.SetFloat("BlinkySpeed", 0.5f);
    PlayerPrefs.SetFloat("PacManSpeed", 0.25f);
    PlayerPrefs.SetInt("PlayerLives", 3);
    PlayerPrefs.SetInt("LivesGained", 1);
    PlayerPrefs.SetFloat("SuperPillPercent", 0.25f);
    PlayerPrefs.SetInt("SuperPillDuration", 10);

    GS.value = PlayerPrefs.GetInt("GridSize");
    WP.value = PlayerPrefs.GetFloat("WallPercent") * 100;
    BS.value = (1 - PlayerPrefs.GetFloat("BlinkySpeed")) / 0.125f;
    PS.value = (1 - PlayerPrefs.GetFloat("PacManSpeed")) / 0.0625f - 9;
    PL.value = PlayerPrefs.GetInt("PlayerLives");
    LG.value = PlayerPrefs.GetInt("LivesGained");
    SPP.value = PlayerPrefs.GetFloat("SuperPillPercent") * 100;
    SPD.value = PlayerPrefs.GetInt("SuperPillDuration");
}
```

Il metodo *ResetValues()* setta tutte le *PlayerPrefs* ai valori di *default* ed attribuisce lo stesso valore pure agli *Slider*.

4.1.7 Menu Manager

Lo script *MenuManager* si occupa di gestire le pagine di menu per navigare le diverse scene.

4.1.7.1 Variabili

```
public GameObject MenuCanvasPrefab;
public static bool isJoystick = false;
```

Ci sono 2 variabili pubbliche, una di tipo *GameObject* per farsi passare una *Prefab* con dentro tutti gli elementi del menu e l'altra è un *bool* statico per sapere se viene usato un *joystick* nella pagina di scelta di modalità di gioco, che di *default* è impostato su falso.

4.1.7.2 Start()

```
void Start()
{
    isJoystick = false;
    var canvas = Instantiate(MenuCanvasPrefab,
        new Vector3(550, 259.5f, 10),
        Quaternion.identity);
    canvas.name = "Canvas";

    var buttons = canvas.GetComponentsInChildren<Transform>();
    foreach (var button in buttons)
    {
        if (button.name == "StartButton")
        {
            button.GetComponent<Button>().onClick.AddListener(IniziaGioco);
        }
        else if (button.name == "SettingsButton")
        {
            button.GetComponent<Button>().onClick.AddListener(GoToGameMode);
        }
        else if (button.name == "ExitButton")
        {
            button.GetComponent<Button>().onClick.AddListener(EsciApplicazione);
        }
    }
}
```

Il metodo *Start()* del *MenuManager* inizia istanziando la *prefab* passata in una variabile di tipo *Canvas* con la sua posizione e poi gli dà il nome "Canvas".

Poi assegna un *Listener* a tutti i bottoni per fare in modo che quando un bottone viene premuto viene eseguito il metodo richiesto.

4.1.7.3 Update()

```
private void Update()
{
    // If X on joystick is pressed then start the game
    if (Input.GetKeyDown("joystick button 1"))
    {
        IniziaGioco();
    }

    // If O on joystick is pressed then exit application
    if (Input.GetKeyDown("joystick button 2"))
    {
        EsciApplicazione();
    }

    // If [] on joystick is pressed then go to game mode page
    if (Input.GetKeyDown("joystick button 0"))
    {
    }
}
```

```
        isJoystick = true;
        GoToGameMode();
    }
}
```

Il metodo *Update()* guarda se vengono premuti i pulsanti sul *controller*. Se viene premuto "X" inizia il gioco, se viene premuto "O" esce dall'applicazione e se viene premuto "[" va alla pagina per cambiare modalità di gioco e viene impostata *true* la variabile *isJoystick*.

4.1.7.4 IniziaGioco()

```
public void IniziaGioco()
{
    SceneManager.LoadScene(1);
}
```

Usa lo *SceneManager* per caricare la scena principale di gioco.

4.1.7.5 ReturnToMenu()

```
public void ReturnToMenu()
{
    SceneManager.LoadScene(0);
}
```

Usa lo *SceneManager* per caricare la scena di menu di gioco.

4.1.7.6 GoToGameMode()

```
public void GoToGameMode()
{
    SceneManager.LoadScene(3);
}
```

Usa lo *SceneManager* per caricare la scena per selezionare la modalità di gioco.

4.1.7.7 EsciApplicazione()

```
public void EsciApplicazione()
{
    Application.Quit();
}
```

Esce dall'applicazione.

4.2 Unity Engine

Unity è un motore grafico multiplatforma che consente lo sviluppo di videogiochi e altri contenuti interattivi, quali visualizzazioni architettoniche o animazioni 3D in tempo reale. Però nel nostro caso lo usiamo per creare un videogioco 2D.

Il linguaggio che viene principalmente usato per programmare in *Unity* è C# (come visto precedentemente negli *script*). Però non basta conoscere la programmazione, ci sono molti altri aspetti da conoscere per usare il *Unity Engine*.

4.2.1 Impostazioni

In *Unity* si possono cambiare le impostazioni dell'*Engine*, per accomodare il lavoro o per funzionalità importanti. Qui riporterò le diverse cose che ho impostato io.

4.2.1.1 External Tools

La prima cosa che ho cambiato era quale applicativo usare per modificare i miei *script*.

Volevo usare *Visual Studio* perché a parte accomodare la scrittura mi dava la possibilità di compilare la soluzione.

Lo si può trovare sotto “*Edit*” → “*Preferences...*”.

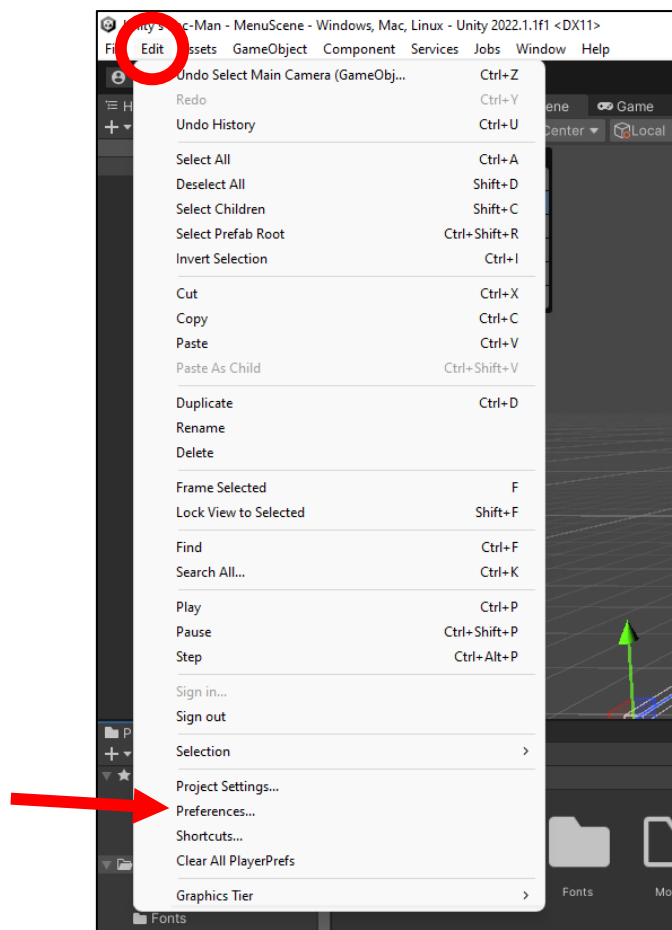


Figura 11 - Unity Preferences

Poi dobbiamo andare su “*External Tools*” e cambiare “*External Script Editor*” all'applicativo che si vuole usare, nel mio caso “*Visual Studio Community 2019*”.

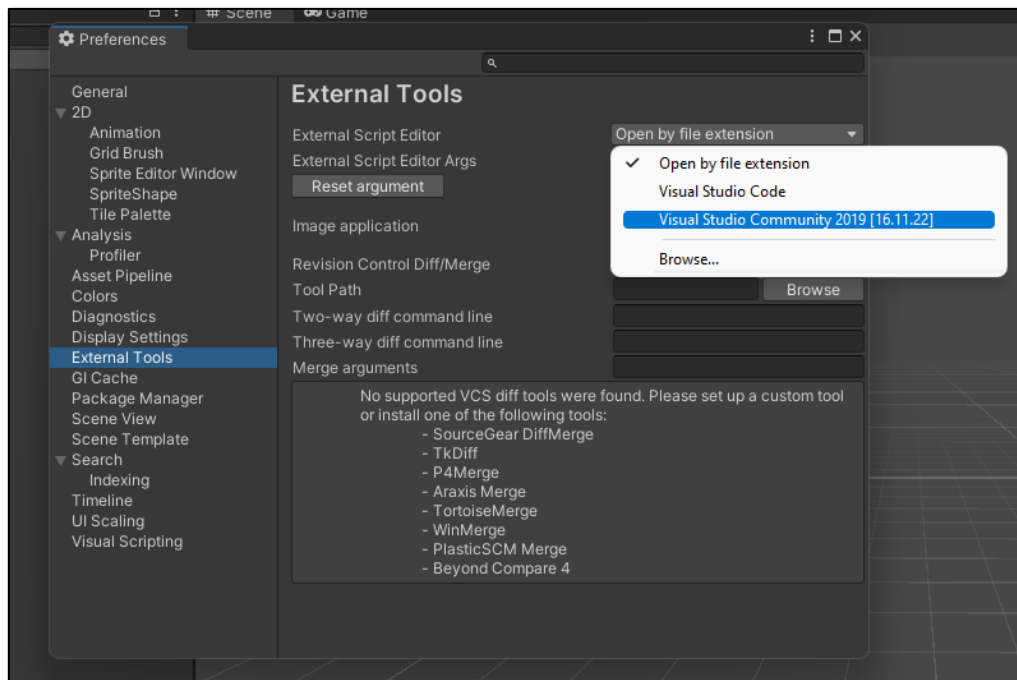


Figura 12 - Unity Preferences External Tools

Ora quando si prova ad aprire uno *script* da *Unity* verrà aperto con *Visual Studio*.

4.2.1.2 Script Execution Order

Quando si avvia il gioco, per essere sicuri che uno *script* parte prima di un altro si deve andare sotto “*Edit*” → “*Project Settings...*”.

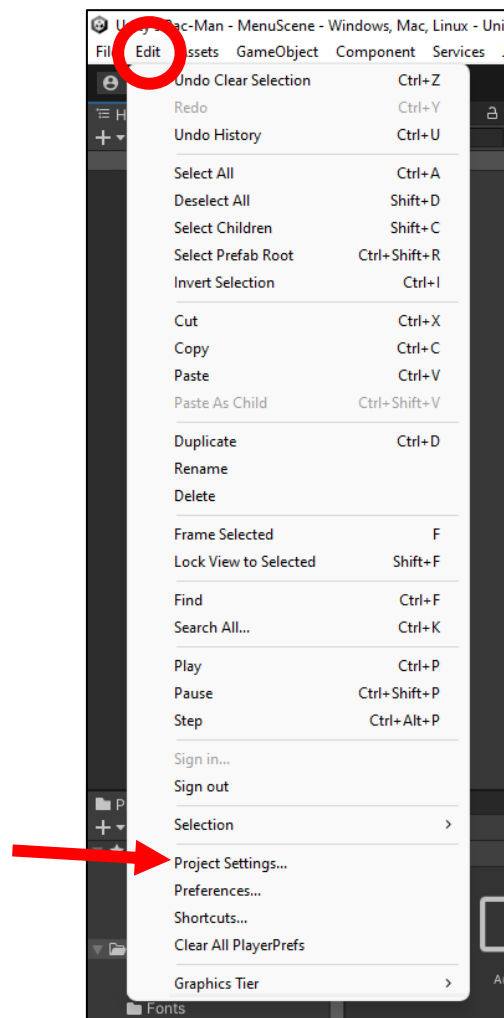


Figura 13 - Unity Project Settings

Poi dobbiamo andare su “*Script Execution Order*” e con il “+” si può aggiungere uno *script*. E poi si può trascinare su e giù gli *script* o cambiare il valore accanto per cambiare l’ordine di esecuzione degli *script*.

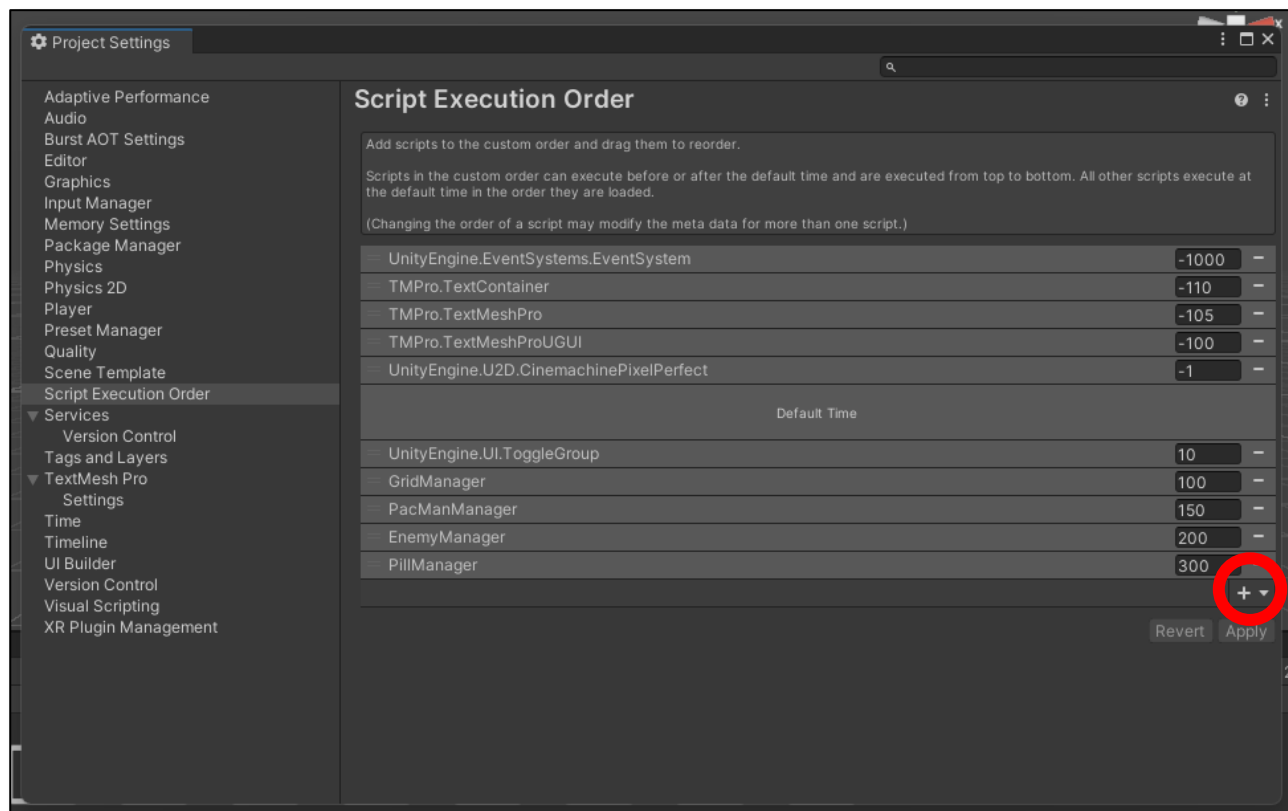


Figura 14 - Unity Script Execution Order

E infine si deve cliccare su “*Apply*” per confermare le modifiche.

4.2.1.3 Input Manager

In *Unity* certi tasti sono già abbinati a delle *axis* di *Unity*, ad esempio la “w” muove in modo positivo l'asse verticale.

Per i tasti con la tastiera non ho dovuto cambiare niente, però essendo che ho aggiunto la possibilità di giocare con un joystick ho dovuto cambiare un paio di cose nell'impostazioni del progetto.

Prima di tutto si deve andare in “*Project Settings*” (vedi precedentemente) e selezionare “*Input Manager*”, da lì dovrebbe apparire una finestra con diverse *axis*. Ma nel nostro caso ci interessano solo le assi “*Horizontal*” e “*Vertical*”. Ci sono doppi di queste *axis* ed è normale.

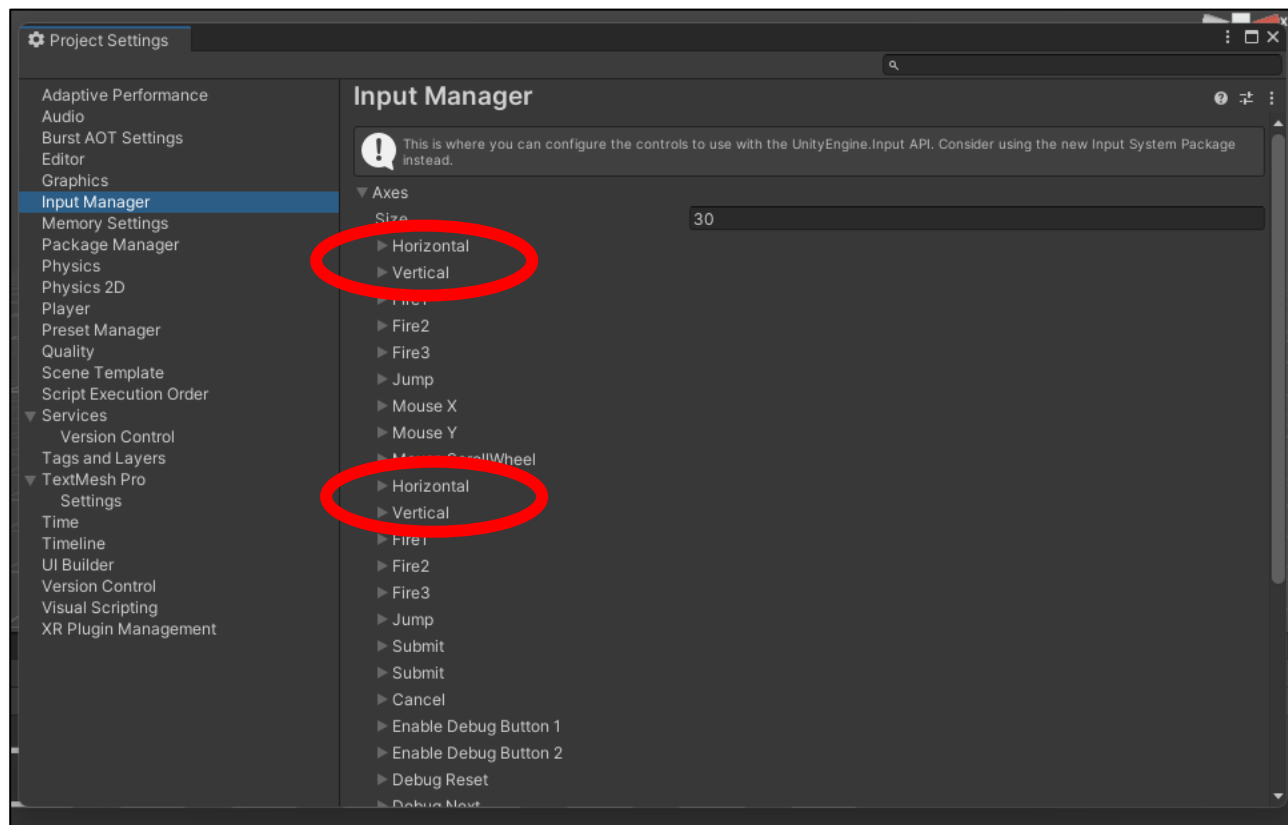


Figura 15 - Unity Input Manager

Nel nostro caso vogliamo avere la tastiera e le freccette sul *joystick* come comandi di gioco. Quindi la prima coppia di *axis* può essere ignorata, mentre nel secondo dobbiamo modificare i valori come nel modo seguente per far sì che l'azione accada subito senza ritardo di segnale. Entrambi *axis* hanno gli stessi valori a parte per il “Name” e “Axis” di cui si deve cambiare per “*Horizontal*” in “*7th axis (Joystick)*” (freccette sinistra e destra) e per “*Vertical*” in “*8th axis (Joystick)*” (freccette su e giù). Vedi immagine seguente.

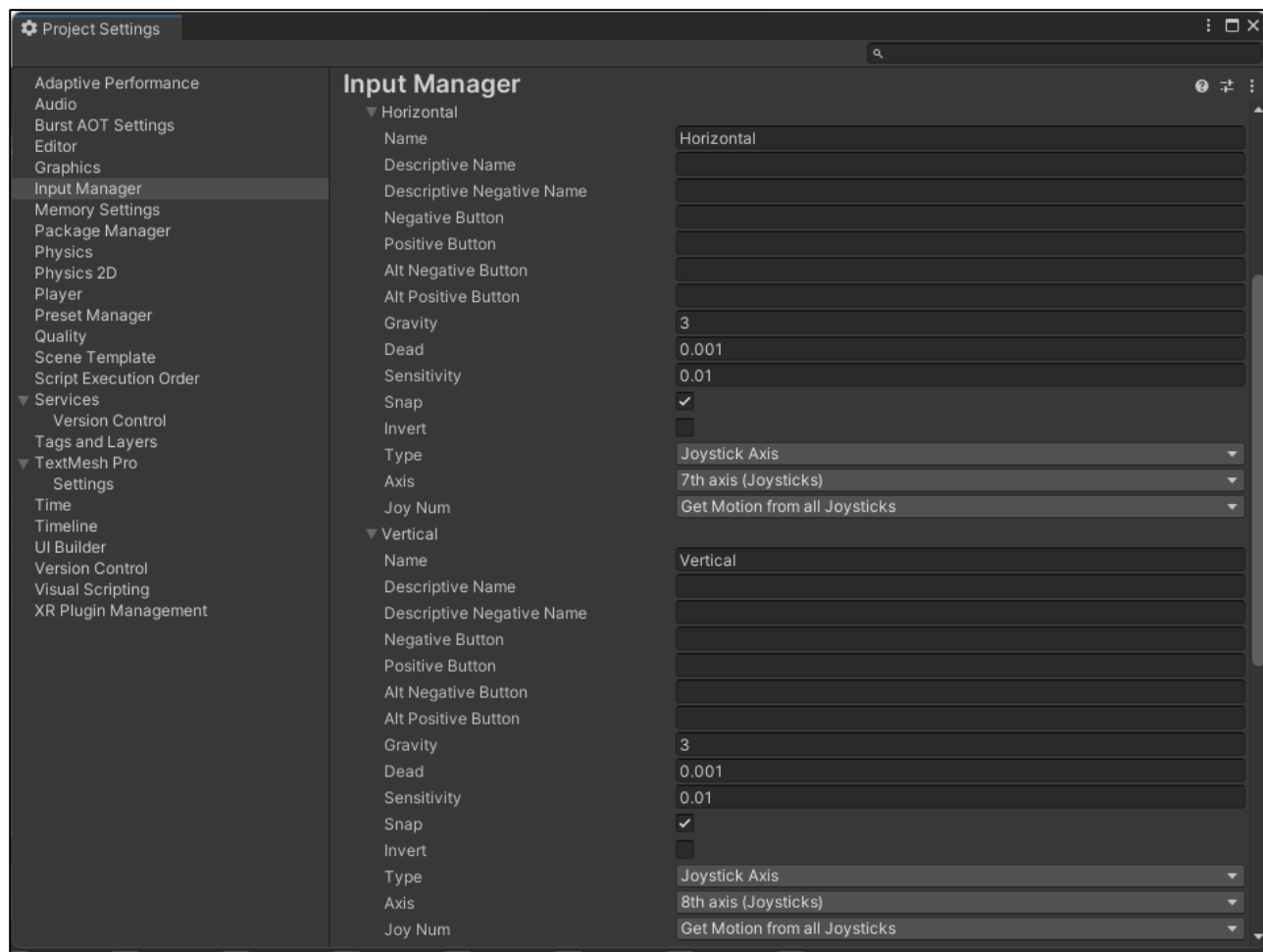


Figura 16 - Unity Input Manager Details

Ora se i tasti della tastiera e/o le freccette del *joystick* venissero premute dovrebbero aumentare o diminuire l'*axis*.

4.2.2 Scene Manager

In *Unity* per un gioco si creano diverse scene per le diverse parti del gioco. Per cambiare scena si deve invocare lo *Scene.Manager* passandoli il numero della scena alla quale si vuole andare. Il numero della scena può essere assegnato nelle *Build Settings* di *Unity*. Per aggiungere la scena basta premere sul bottone “*Add Open Scenes*”. La scena in cima ha l'indice 0 e poi le scene sotto aumentano di uno. Per invertire l'ordine delle scene basta trascinarle in basso od in alto a dipendenza di che indice si vuole.

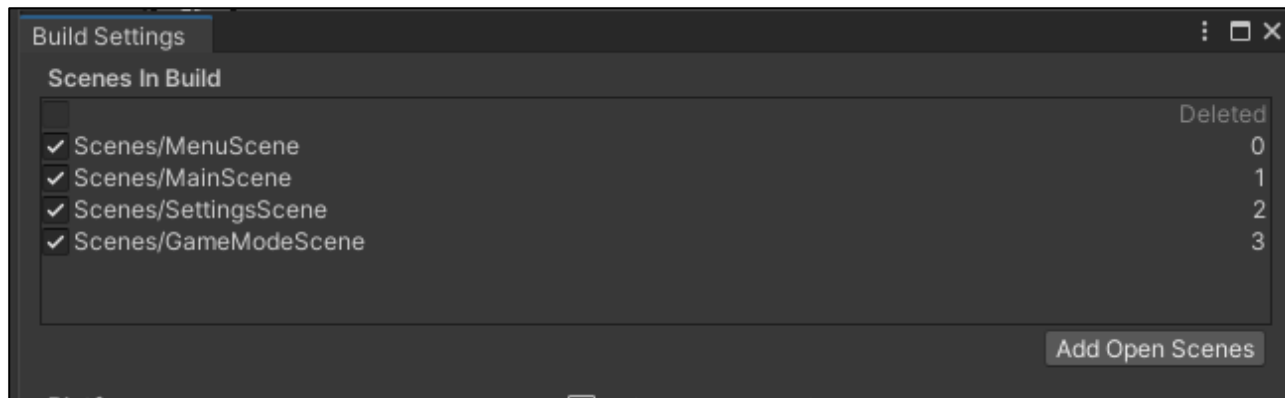


Figura 17 - SceneManager

4.2.3 Prefab

Per creare una *prefab* basta premere click destro nella cartella in cui si vuole creare la *prefab* e poi selezionare “Create” → “Prefab”.

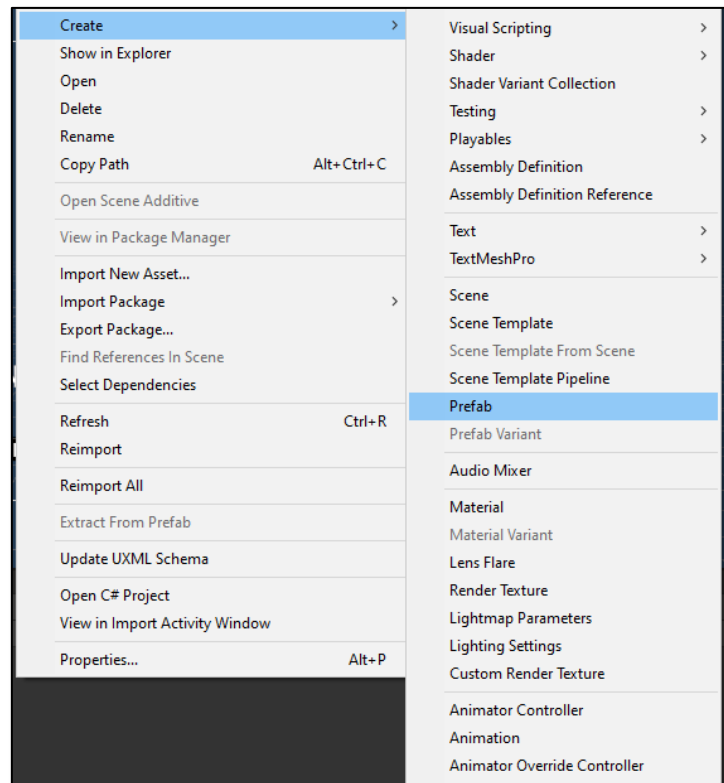


Figura 18 - Crea Prefab

Per le *prefab* che servano come interfaccia di menu ho aggiunto il componente “Canvas” dove ho poi aggiunto tutti gli oggetti che servono per l’interfaccia.

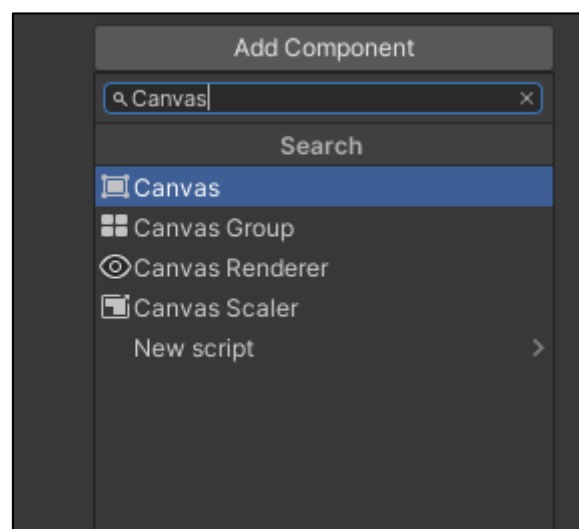


Figura 19 - Aggiungi Canvas

Per le *prefab* che servono per le *Sprite* ho aggiunto il componente “*Sprite Renderer*”.

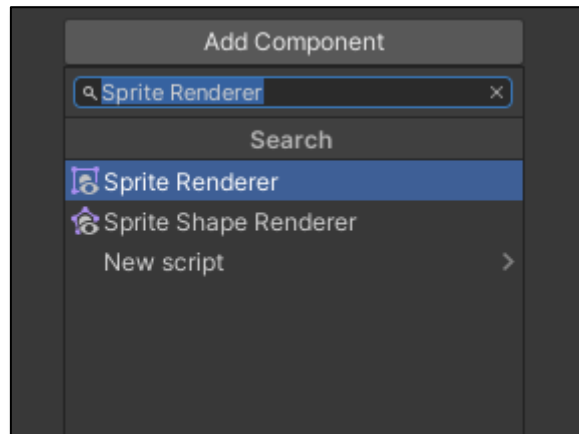


Figura 20 - Aggiungi SpriteRenderer

Nel caso del fantasma ho dovuto anche aggiungere i componenti “*Animator*” ed “*Animation*” per poter rendere il fantasma animato. Nell’*Animator* ho aggiunto il *controller* e nell’*Animation* ho aggiunto la direzione iniziale che poi viene cambiata dall’*Animator* a dipendenza di in che direzione sta andando.

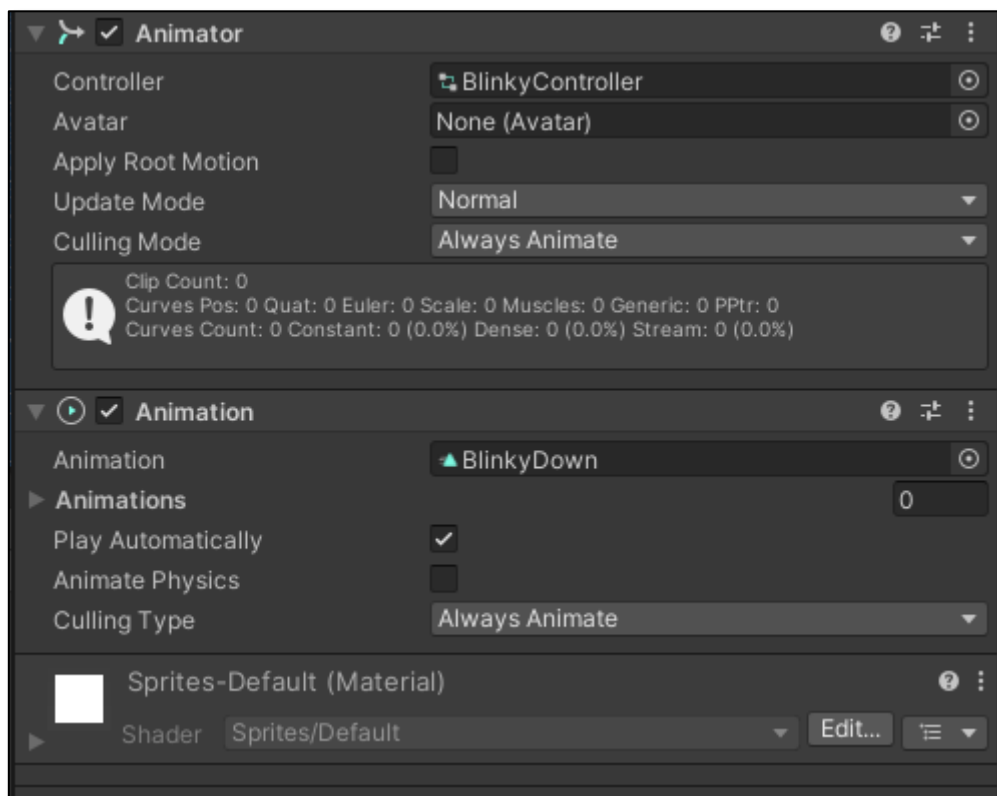


Figura 21 - Animator ed Animation

4.2.4 Animazioni

Per dare della vivenza al nostro gioco, bisogna aggiungere delle animazioni: Per il *Pac-Man* lo abbiamo fatto attraverso lo *script* dicendogli di cambiare il suo immagine ogni x secondi. Però per il fantasma ho usato le funzionalità che *Unity* ci propone.

4.2.4.1 Animator Controller

Per prima cosa dobbiamo creare un *Animator Controller* (non da essere confusi con il *joystick controller*), che serve per tenere tutte le animazioni di un “personaggio”.

Per farlo si deve fare tasto destro con il *mouse* e poi andare sotto “*Create*” e scegliere “*Animator Controller*”

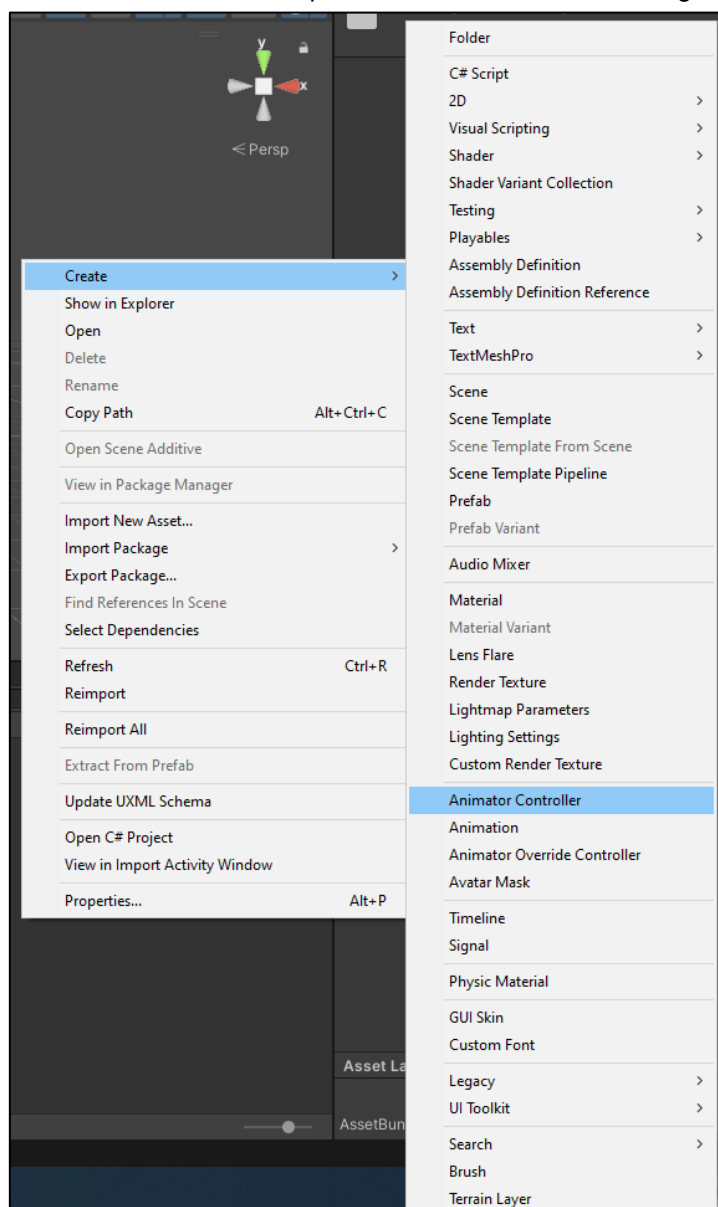


Figura 22 - Unity Create Animator Controller

Prima di tutto dobbiamo aggiungere uno stato di animazione di *default*. Per farlo si fa tasto destro e cliccare su “*Create State*” → “*From New Blend Tree*”.

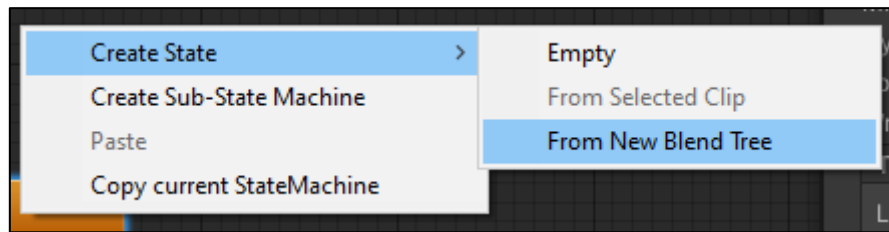


Figura 23 - Unity Animator Idle

Dal *Blend Tree* aggiungiamo i diversi stati di animazione. Per creare le diverse animazioni e assegnarle dentro l'*Animator Controller* bisogna fare tasto destro e cliccare su "Create State" → "Empty".

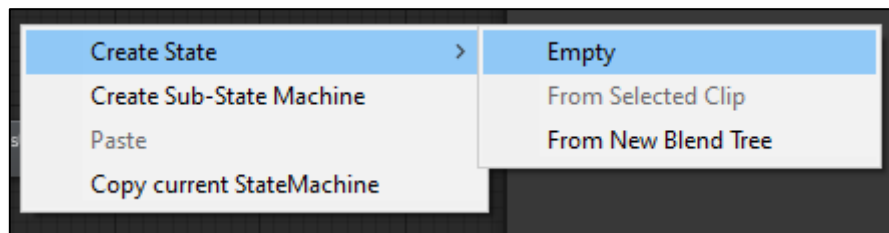


Figura 24 - Unity Animator State

Poi di nuovo tasto destro, però sopra lo stato di cui si vuole aggiungere, e cliccare su "Make Transition".

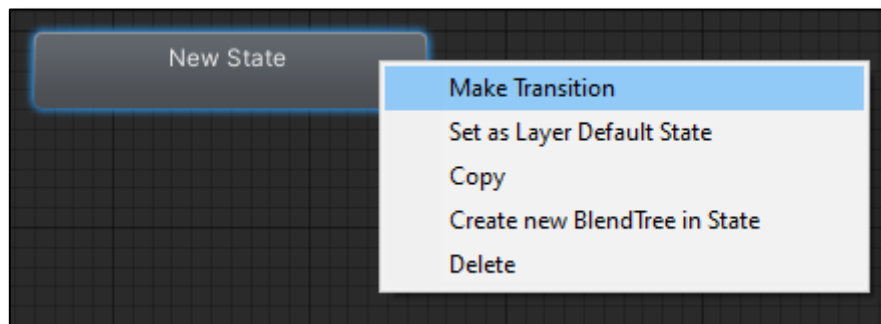


Figura 25 - Unity Animator Make Transition

Poi bisogna trascinare il cursore sopra lo stato del *Blend Tree* e poi si deve creare un'altra *Transition* all'incontrario.

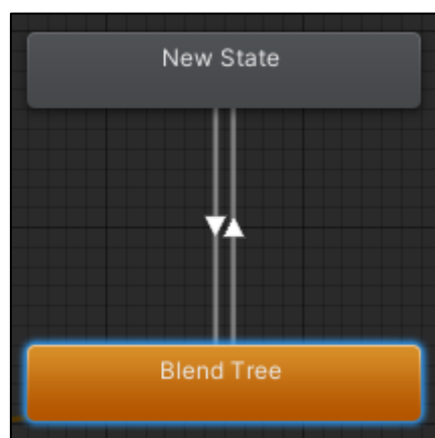


Figura 26 - Unity Animator Transition

In questo caso questo stato lo uso quando il fantasma si muove in su, però è uguale per tutte le direzioni.

Prima di tutto dobbiamo creare un parametro per specificare quando il fantasma si muoverà in su, l'animazione dovrà cambiare. Si può fare cliccando su “Parameters” e poi con il “+” e scegliere “Bool” e poi dargli un nome logico ad esempio “IsUp”.

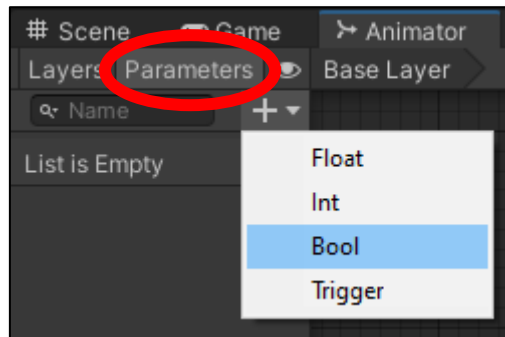


Figura 27 - Unity Animator Parameters

Potete cancellare il parametro “Blend” che non serve nel nostro caso.

Sulla *Transition* che punta verso il *New State* dobbiamo aggiungere una condizione che è collegato con il nostro *IsUp*. Poi bisogna disabilitare “Has Exit Time” e mettere tutti i valori a 0 perché altrimenti ci sarà un leggerlo ritardo con l'animazione che non è bello da vedere.

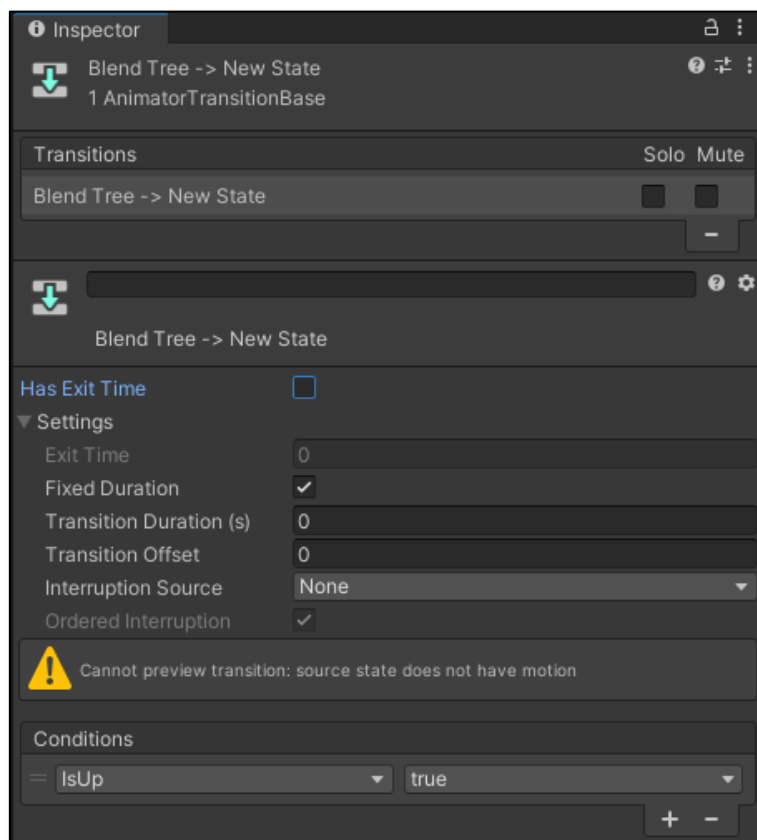


Figura 28 - Unity Animator Transition True

L'errore che appare è normale essendo che non è ancora stato aggiunto l'animazione.

Poi si deve fare la stessa cosa per il la *Transition* che punta verso il basso. Però con la condizione a *false*. Questa ci permetterà di cambiare stato di animazione in modo semplice con lo *script*.

4.2.4.2 Animation

Ora dobbiamo creare ed assegnare un'animazione ad uno stato dentro l'*Animator Controller*. Per crearlo si deve fare tasto destro e selezionare "Create" → "Animation".

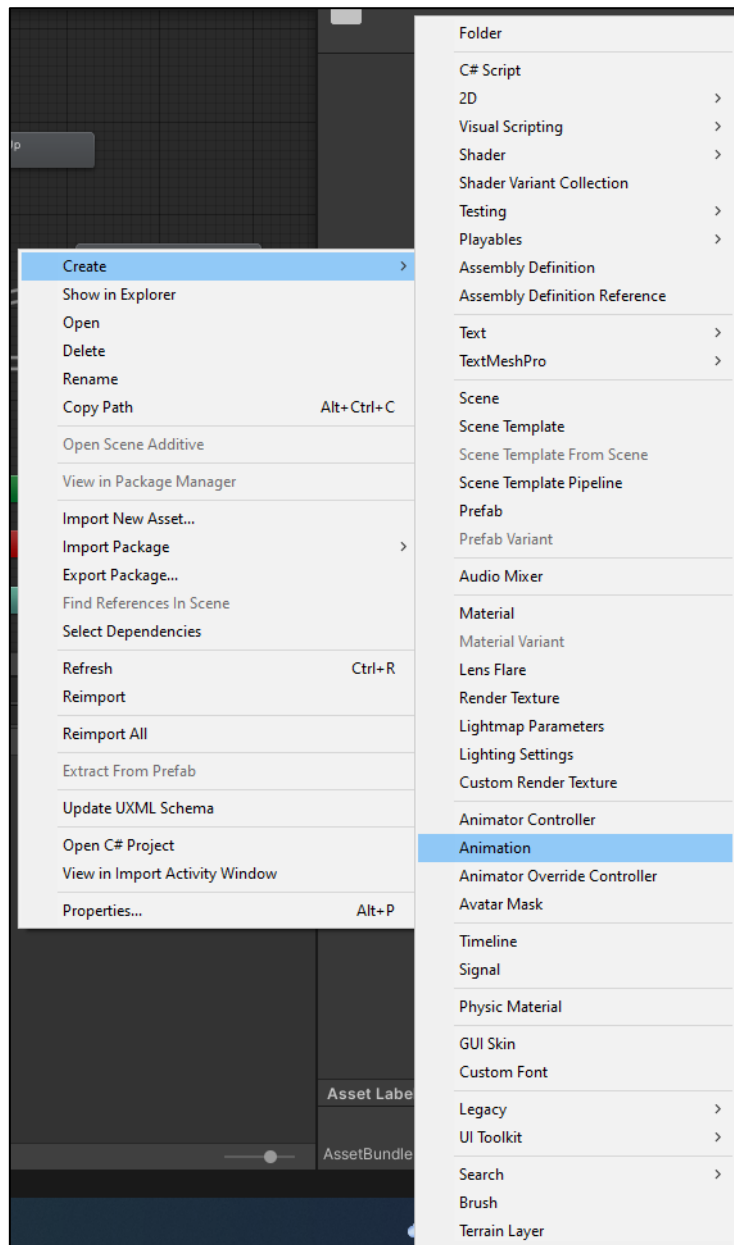


Figura 29 - Unity Create Animation

Non si potrà ancora modificare l'animazione finché non è connesso ad un'*Animator Controller* che a sua volta è connesso ad un *prefab*.

Per aggiungere l'animazione all'*Animator Controller* è abbastanza semplice. Devi prendere e trascinare l'animazione dentro il riquadro di "New State" → "Motion".

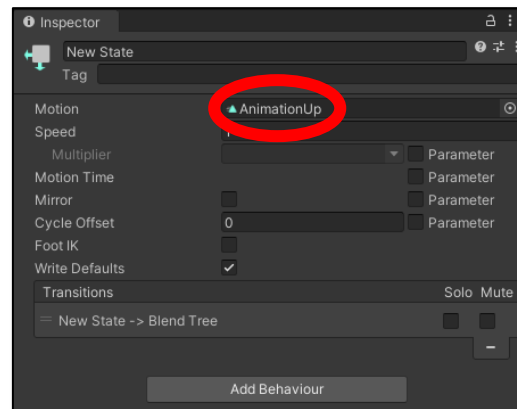


Figura 30 - Unity Animation Connect

Per poi aggiungerlo al *prefab* basta selezionare il *prefab* e aggiungere una sezione *Animator* (se non già fatto) e trascinare l'*Animator Controller* dentro "Animator" → "Controller".

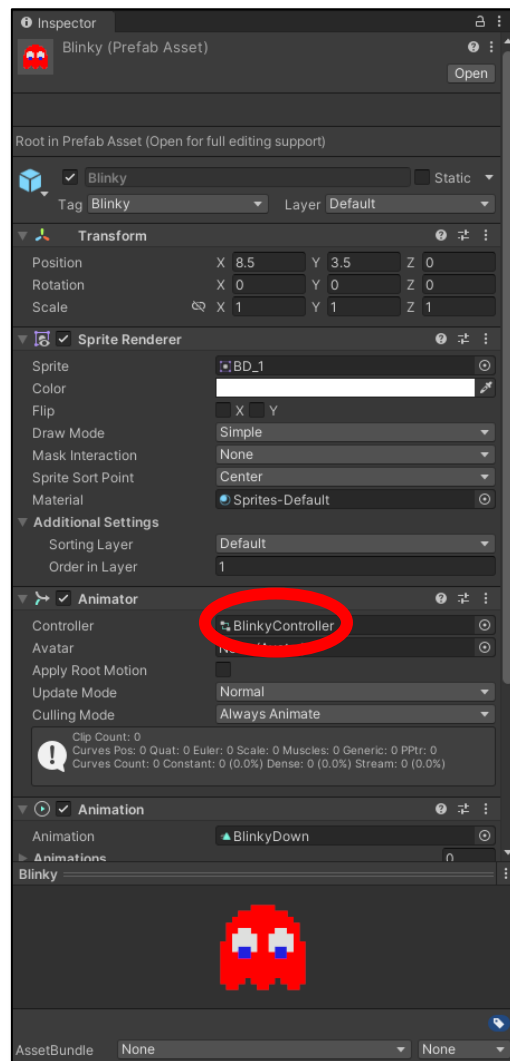


Figura 31 - Unity Animation Prefab

Ora si può aprire l'animazione, però ti lascerà modificare finché non selezioni di nuovo il *prefab*. Dopo di che basta trascinare le immagini che si vogliono usare per l'animazione dentro la *timeline*.

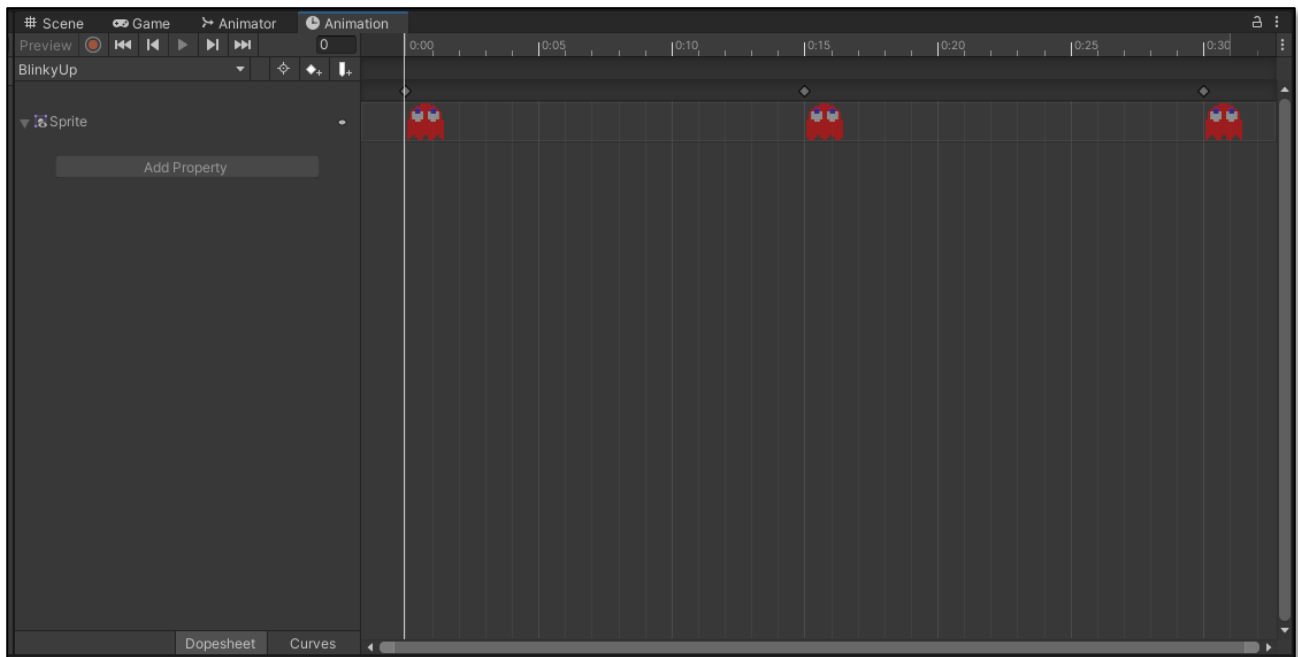


Figura 32 - Unity Animation Timeline

Ora l'animazione è completata per il fantasma quando guarda in su. Poi ho rifatto questi passaggi per ogni diverso stato di animazione e rinominato i nomi degli stati per ordinazione, ed alla fine l'*Animator Controller* mi è uscito così.

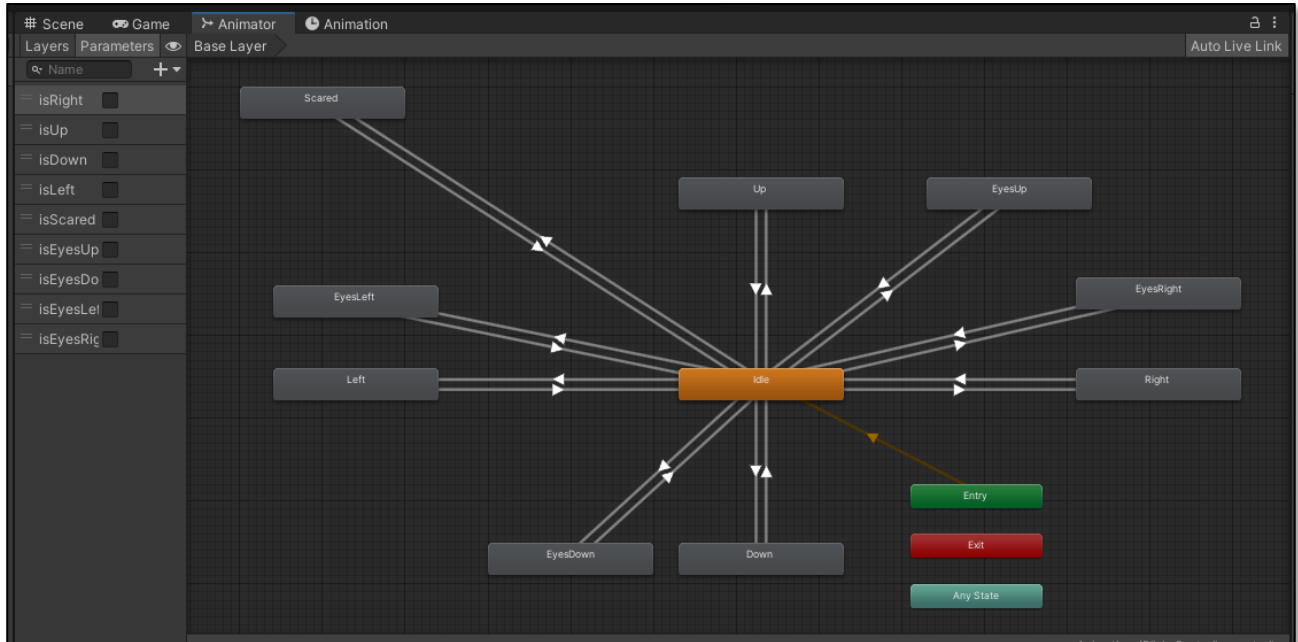


Figura 33 - Unity Animator Controller Final

4.2.5 In-Game GUI

Per fare le *GUI* ho creato delle *prefab* ed ho messo dentro i testi, bottoni e *Slider* di cui avevo bisogno per le interfacce. Per aggiungere gli oggetti ho fatto click destro sulla *prefab*, sono andato sotto *UI* e poi ho selezionato l'oggetto di cui ho bisogno.

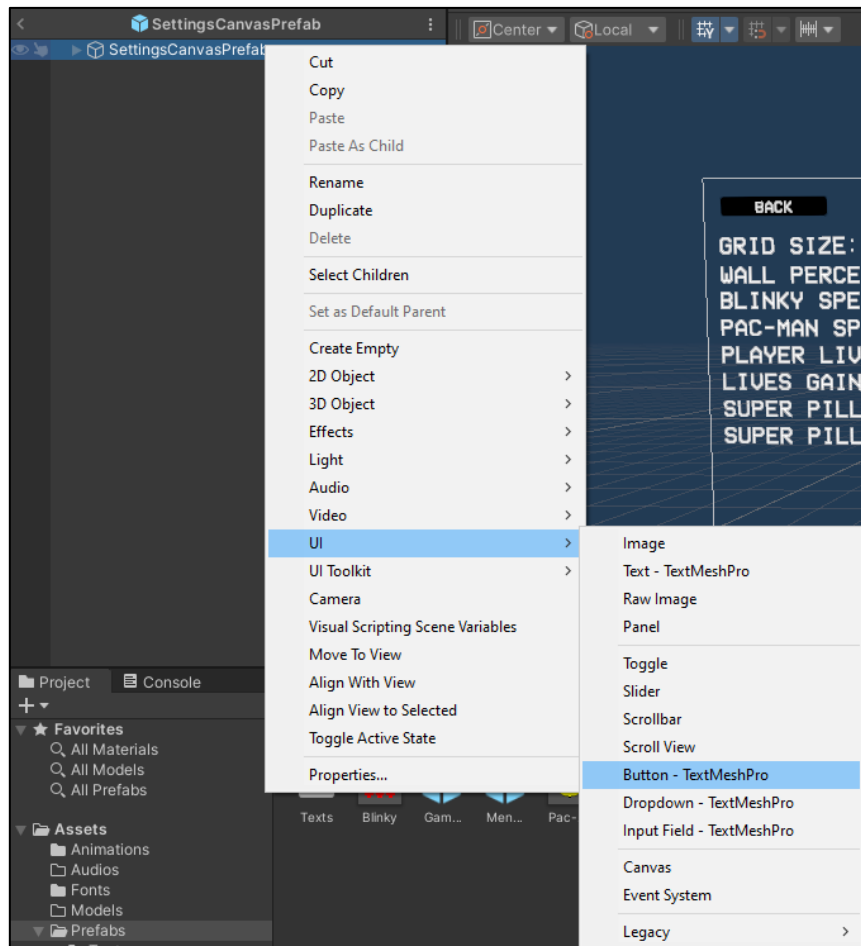


Figura 34 - Unity UI

Per la posizione dell'oggetto ho usato il componente “*Rect Transform*” per potergli assegnare la posizione esatta di cui ho bisogno sull'asse x e y, e per modificare la grandezza e l'altezza.

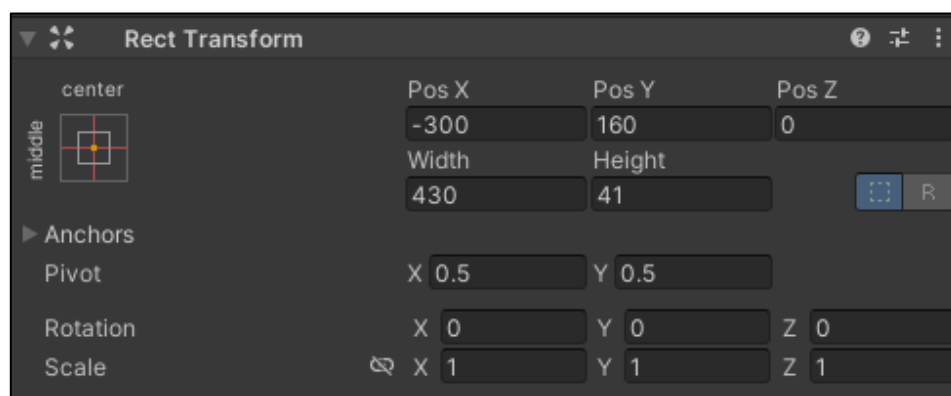


Figura 35 - Rect Transform

Ho piazzato gli *Slider* nello stesso modo dei bottoni ed ho messo su *true* la variabile per poter selezionare solo numeri interi. Il range dei numeri viene poi settato nel codice.

Nelle *TextMeshPro* dentro i bottoni o per i testi ho messo un font retro per farlo sembrare più simile al gioco originale ed ho modificato il colore del testo in bianco. Se il testo non viene aggiornato dall'update allora ho scritto cosa deve apparire sul *TextMesh* nella *TextBox*.

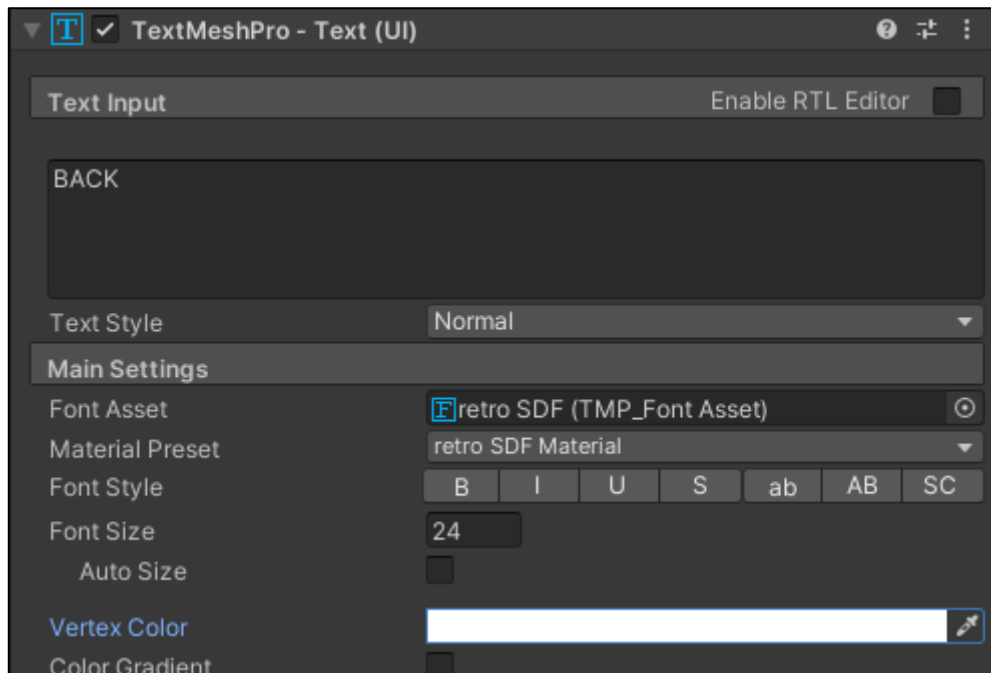


Figura 36 - TextMeshPro

Per rendere i bottoni e gli *Slider* interattivi dall'utente si deve aggiungere un *EventSystem* che è responsabile di processare gli eventi nella scena e dentro la *Canvas* della *prefab* deve esserci il componente "*Graphic RayCaster*" che guarda se viene colpito uno degli oggetti grafici sul *Canvas*.

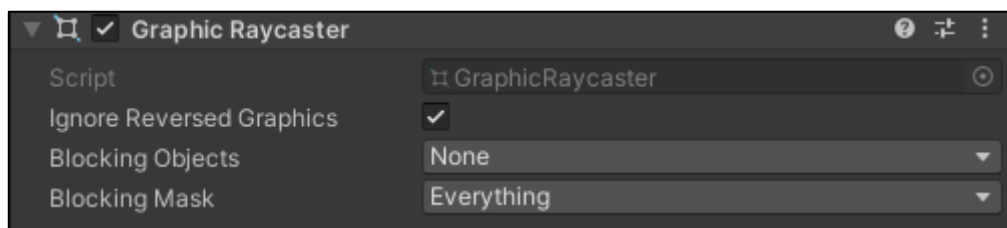


Figura 37 - Graphic Raycaster

4.2.6 Build and Run

Quando si è sicuri di volere creare una prima *Build* si può andare sotto “File” → “Build Settings...”.

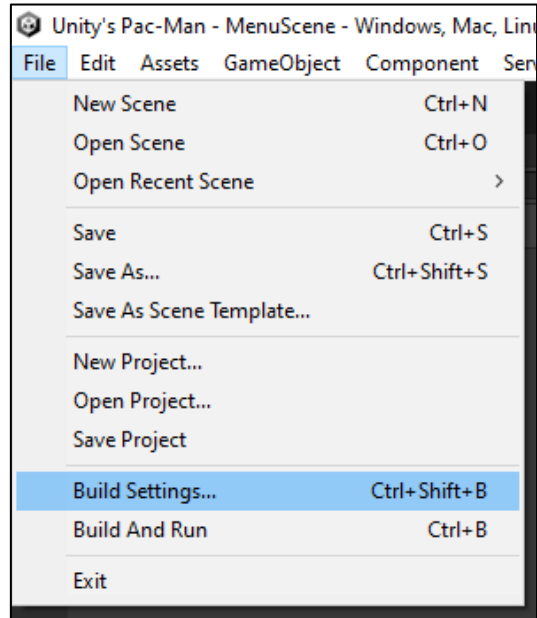


Figura 38 - Unity Find Build Settings

Da qui si può scegliere che tipo di esportazione, nel mio caso un applicativo per *Windows*. E infine preme su “Build”.

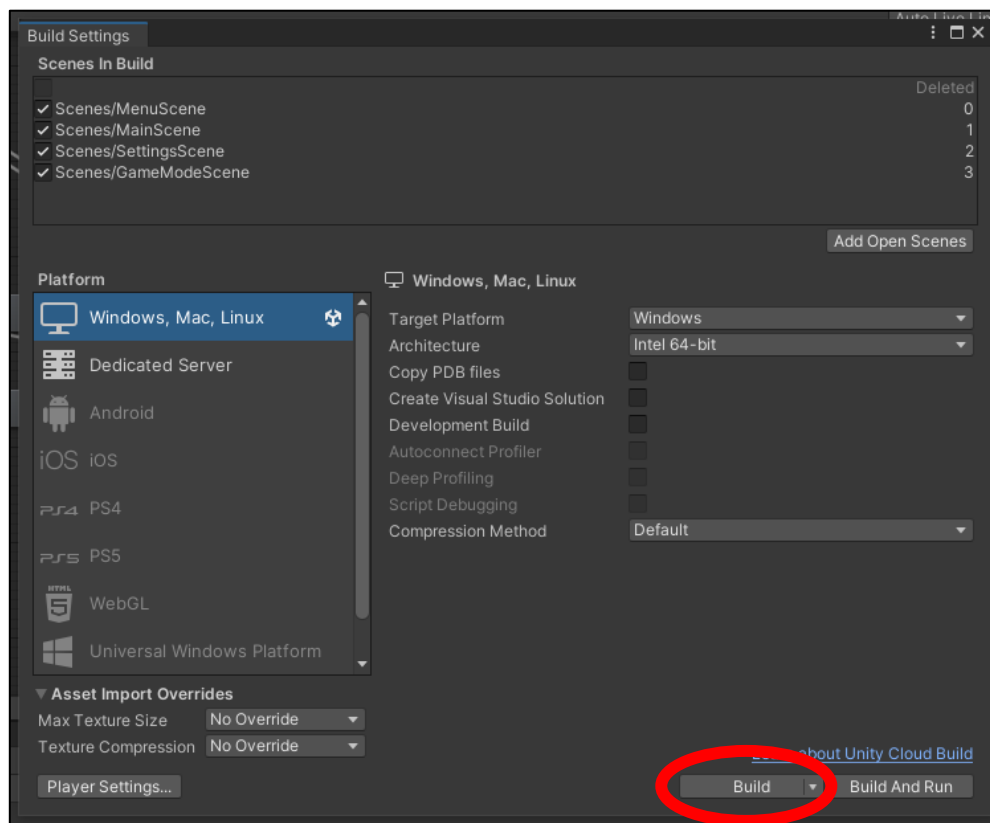


Figura 39 - Unity Build Settings

Verrà creato una cartella *Build* con diversi file, ma quello che a noi interessa è l'applicativo del gioco che viene creato con il suo nome e con l'estensione “.exe”.

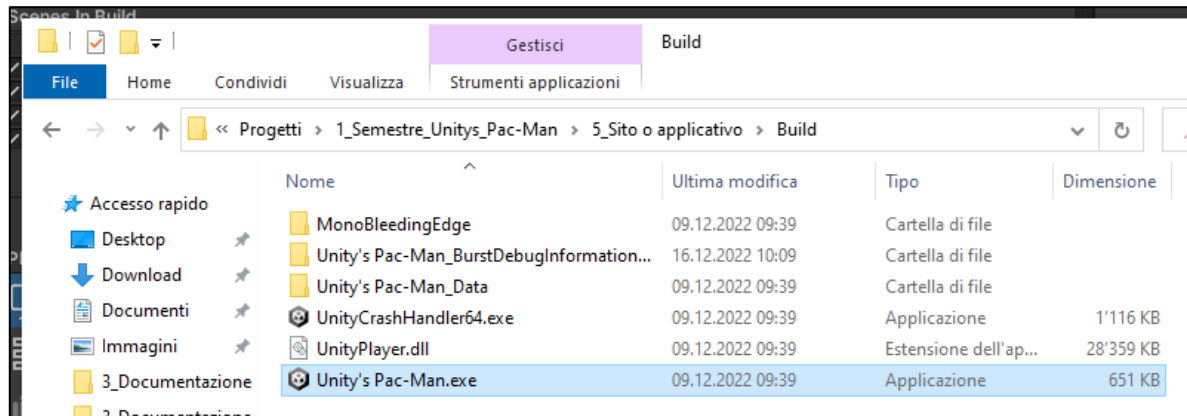


Figura 40 - Unity Applicazione Finale

Basta eseguire quell'applicativo e il gioco partirà in schermo intero e potrai goderti l'esperienza.

5 Test

5.1 Protocollo di test

Definire in modo accurato tutti i test che devono essere realizzati per garantire l'adempimento delle richieste formulate nei requisiti. I test fungono da garanzia di qualità del prodotto. Ogni test deve essere ripetibile alle stesse condizioni.

Test Case:	TC-001	Nome:	Mappa Generata Casualmente
Riferimento:	REQ-01		
Descrizione:	Verrà generata una mappa senza interruzioni tra il percorso.		
Procedura:	<ol style="list-style-type: none"> 1. Fare partire l'applicativo in Unity. 2. Avviare una partita. 3. Controllare che la mappa sia stata generata correttamente. 		
Risultati attesi:	Una mappa casuale senza punti irraggiungibile in cui le mura bloccano il passaggio.		

Test Case:	TC-002	Nome:	Movimento Pac-Man
Riferimento:	REQ-02		
Descrizione:	Muovere Pac-Man usando i tasti sulla tastiera.		
Procedura:	<ol style="list-style-type: none"> 1. Fare partire L'applicativo in Unity. 2. Avviare una partita. 3. Usare i diversi tasti seguenti per fare muovere Pac-Man, in diverse direzioni, di un blocco sulla mappa: <ul style="list-style-type: none"> • Tasto "w" o freccia in su: sposta in su. • Tasto "a" o freccia a sinistra: sposta a sinistra. • Tasto "s" o freccia in giù: sposta in giù. • Tasto "d" o freccia a destra: sposta a destra. 		
Risultati attesi:	Il Pac-Man si muove in 4 direzioni usando i tasti specificati precedentemente.		

Test Case:	TC-003	Nome:	Movimento Pac-Man Attraverso il Labirinto
Riferimento:	REQ-02		
Descrizione:	Muovere Pac-Man dentro la mappa.		
Procedura:	<ol style="list-style-type: none"> 1. Fare partire L'applicativo in Unity. 2. Avviare una partita. 3. Usare i diversi tasti seguenti per fare muovere Pac-Man, in diverse direzioni, di un blocco sulla mappa: <ul style="list-style-type: none"> • Tasto "w" o freccia in su: sposta in su. • Tasto "a" o freccia a sinistra: sposta a sinistra. • Tasto "s" o freccia in giù: sposta in giù. • Tasto "d" o freccia a destra: sposta a destra. 4. Andare contro un muro in tutte le 4 direzioni 		
Risultati attesi:	Pac-Man si muove in 4 direzioni usando i tasti specificati precedentemente, però non può entrare nei muri della mappa.		

Test Case:	TC-004	Nome:	Piazzamento Pillole nella Mappa
Riferimento:	REQ-03		
Descrizione:	Vengono piazzate pillole su ogni cella della mappa.		
Procedura:	<ol style="list-style-type: none"> 1. Fare partire L'applicativo in Unity. 2. Avviare una partita. 3. Controllare la mappa 		
Risultati attesi:	Le pillole appaiono solo nelle celle della mappa libere e non dentro le mura.		

Test Case:	TC-005	Nome:	Consumazione delle pillole da Pac-Man
Riferimento:	REQ-03		
Descrizione:	Pac-Man consuma le pillole quando sale sulla loro cella.		
Procedura:	<ol style="list-style-type: none"> 1. Fare partire L'applicativo in Unity. 2. Avviare una partita. 3. Usare i diversi tasti seguenti per fare muovere Pac-Man, in diverse direzioni, di un blocco sulla mappa: <ul style="list-style-type: none"> • Tasto "w" o freccia in su: sposta in su. • Tasto "a" o freccia a sinistra: sposta a sinistra. • Tasto "s" o freccia in giù: sposta in giù. • Tasto "d" o freccia a destra: sposta a destra. 4. Mangiare qualche pillola 		
Risultati attesi:	Quando Pac-Man sale su una cella con una pillola e poi cambia cella la cella di prima sarà vuota.		

Test Case:	TC-006	Nome:	Avvicinamento del Fantasma AI al Pac-Man Statico.
Riferimento:	REQ-04		
Descrizione:	Il fantasma si avvicina, blocco per blocco, al giocatore Pac-Man fermo.		
Procedura:	<ol style="list-style-type: none"> 1. Fare partire L'applicativo in Unity. 2. Avviare una partita. 3. Controllare che il fantasma si avvicina di un blocco ogni x secondi (variabile) al Pac-Man. 		
Risultati attesi:	Il fantasma si avvicina a Pac-Man fermo blocco per blocco ogni x secondi.		

Test Case:	TC-007	Nome:	Avvicinamento del Fantasma AI al Pac-Man Dinamico
Riferimento:	REQ-04		
Descrizione:	Il fantasma si avvicina, blocco per blocco, al giocatore Pac-Man che si muove.		
Procedura:	<ol style="list-style-type: none"> 1. Fare partire L'applicativo in Unity. 2. Avviare una partita. 3. Usare i diversi tasti seguenti per fare muovere Pac-Man, in diverse direzioni, di un blocco sulla mappa: <ul style="list-style-type: none"> • Tasto "w" o freccia in su: sposta in su. • Tasto "a" o freccia a sinistra: sposta a sinistra. • Tasto "s" o freccia in giù: sposta in giù. • Tasto "d" o freccia a destra: sposta a destra. 4. Controllare che il fantasma si avvicina di un blocco ogni x secondi (variabile) al Pac-Man. 		
Risultati attesi:	Il fantasma si avvicina a Pac-Man che si sta spostando blocco per blocco ogni x secondi.		

Test Case:	TC-008	Nome:	Avvicinamento del Fantasma AI al Pac-Man dentro le mura della mappa.
Riferimento:	REQ-04		
Descrizione:	Il fantasma si avvicina, blocco per blocco, al giocatore Pac-Man che si muove attraversando i percorsi del labirinto.		
Procedura:	<ol style="list-style-type: none"> 1. Fare partire L'applicativo in Unity. 2. Avviare una partita. 3. Controllare che il fantasma si avvicina blocco per blocco ogni x secondi (variabile) al Pac-Man senza uscire dalla mappa o scontrandosi contro le mura. 		
Risultati attesi:	Il fantasma si avvicina a Pac-Man che si sta spostando blocco per blocco ogni x secondi percorrendo il labirinto senza passare sopra le mura.		

Test Case:	TC-009	Nome:	Consumazione della Super-Pillola Invoca l'invincibilità.
Riferimento:	REQ-05		
Descrizione:	Le Super-Pillole dopo che sono consumate dal Pac-Man gli danno l'invincibilità' contro il fantasma per x secondi (variabile).		
Procedura:	<ol style="list-style-type: none"> 1. Fare partire L'applicativo in Unity. 2. Avviare una partita. 3. Usare i diversi tasti seguenti per fare muovere Pac-Man, in diverse direzioni, di un blocco sulla mappa: <ul style="list-style-type: none"> • Tasto "w" o freccia in su: sposta in su. • Tasto "a" o freccia a sinistra: sposta a sinistra. • Tasto "s" o freccia in giù: sposta in giù. • Tasto "d" o freccia a destra: sposta a destra. 4. Salire su un blocco con una Super-Pillola. 		
Risultati attesi:	Pac-Man diventa invincibile per x secondi.		

Test Case:	TC-010	Nome:	Consumazione della Super-Pillola vittimizza il fantasma.
Riferimento:	REQ-05		
Descrizione:	Le Super-Pillole dopo che sono consumate dal Pac-Man fanno scappare il fantasma per x secondi (variabile).		
Procedura:	<ol style="list-style-type: none"> 1. Fare partire L'applicativo in Unity. 2. Avviare una partita. 3. Usare i diversi tasti seguenti per fare muovere Pac-Man, in diverse direzioni, di un blocco sulla mappa: <ul style="list-style-type: none"> • Tasto "w" o freccia in su: sposta in su. • Tasto "a" o freccia a sinistra: sposta a sinistra. • Tasto "s" o freccia in giù: sposta in giù. • Tasto "d" o freccia a destra: sposta a destra. 4. Salire su un blocco con una Super-Pillola. 		
Risultati attesi:	Il fantasma inizia a scappare da Pac-Man senza uscire dalla mappa o scontrarsi contro le mura.		

Test Case:	TC-011	Nome:	Conteggio punteggio.
Riferimento:	REQ-06		
Descrizione:	Ogni volta che una pillola viene consumata il punteggio mostrato accanto incrementa di uno.		
Procedura:	<ol style="list-style-type: none"> 1. Fare partire L'applicativo in Unity. 2. Avviare una partita. 3. Usare i diversi tasti seguenti per fare muovere Pac-Man, in diverse direzioni, di un blocco sulla mappa: <ul style="list-style-type: none"> • Tasto "w" o freccia in su: sposta in su. • Tasto "a" o freccia a sinistra: sposta a sinistra. • Tasto "s" o freccia in giù: sposta in giù. • Tasto "d" o freccia a destra: sposta a destra. 4. Salire su un blocco con una pillola. 		
Risultati attesi:	Il punteggio mostrato accanto incrementa di un punto.		

Test Case:	TC-012	Nome:	Salvataggio punteggio.
Riferimento:	REQ-06		
Descrizione:	Se il punteggio ottenuto è superiore all'highscore viene salvato come nuovo punteggio massimo.		
Procedura:	<ol style="list-style-type: none"> 1. Fare partire L'applicativo in Unity. 2. Avviare una partita. 3. Usare i diversi tasti seguenti per fare muovere Pac-Man, in diverse direzioni, di un blocco sulla mappa: <ul style="list-style-type: none"> • Tasto "w" o freccia in su: sposta in su. • Tasto "a" o freccia a sinistra: sposta a sinistra. • Tasto "s" o freccia in giù: sposta in giù. • Tasto "d" o freccia a destra: sposta a destra. 4. Salire su un blocco con una pillola. 5. Farsi prendere dal fantasma fino a quando si finiscono le vite 		
Risultati attesi:	Viene salvato e mostrato il punteggio migliore del giocatore dopo che perde.		

Test Case:	TC-013	Nome:	Vittoria.
Riferimento:	REQ-07		
Descrizione:	Quando tutte le pillole sono consumate, una nuova mappa viene generata dove i personaggi vengono ripristinati e il giocatore ottiene x vite (variabile).		
Procedura:	<ol style="list-style-type: none"> 1. Fare partire L'applicativo in Unity. 2. Avviare una partita. 3. Usare i diversi tasti seguenti per fare muovere Pac-Man, in diverse direzioni, di un blocco sulla mappa: <ul style="list-style-type: none"> • Tasto "w" o freccia in su: sposta in su. • Tasto "a" o freccia a sinistra: sposta a sinistra. • Tasto "s" o freccia in giù: sposta in giù. • Tasto "d" o freccia a destra: sposta a destra. 4. Salire su tutti i blocchi della mappa con su una pillola. 		
Risultati attesi:	Una nuova mappa viene generata dove i personaggi vengono ripristinati e il giocatore ottiene x vite (variabile).		

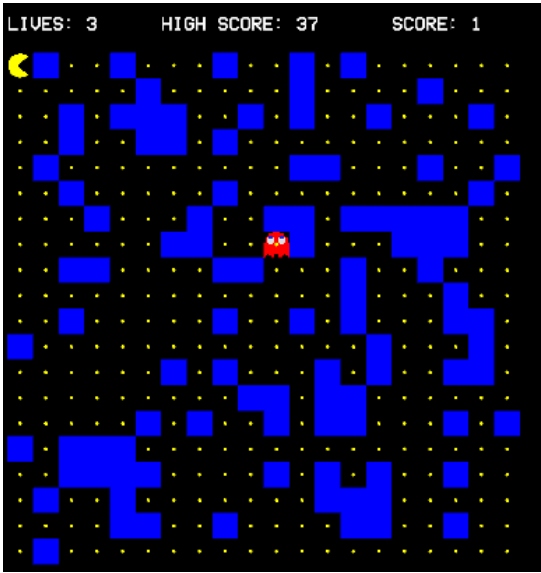
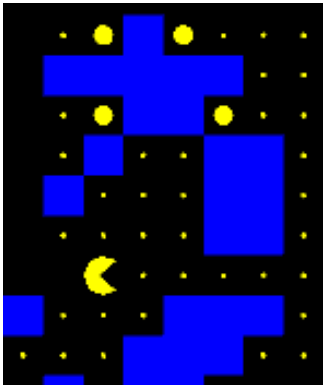
Test Case:	TC-014	Nome:	Perdita vita.
Riferimento:	REQ-07		
Descrizione:	Quando Pac-Man viene preso dal fantasma perde una vita se ne ha ancora e viene ricaricato nel punto di partenza.		
Procedura:	<ol style="list-style-type: none"> 1. Fare partire L'applicativo in Unity. 2. Avviare una partita. 3. Usare i diversi tasti seguenti per fare muovere Pac-Man, in diverse direzioni, di un blocco sulla mappa: <ul style="list-style-type: none"> • Tasto "w" o freccia in su: sposta in su. • Tasto "a" o freccia a sinistra: sposta a sinistra. • Tasto "s" o freccia in giù: sposta in giù. • Tasto "d" o freccia a destra: sposta a destra. 4. Farsi prendere dal fantasma. 		
Risultati attesi:	Il conteggio delle vite mostrato sulla pagina diminuisce di uno e Pac-Man ed il fantasma vengono ricaricati al punto di partenza.		


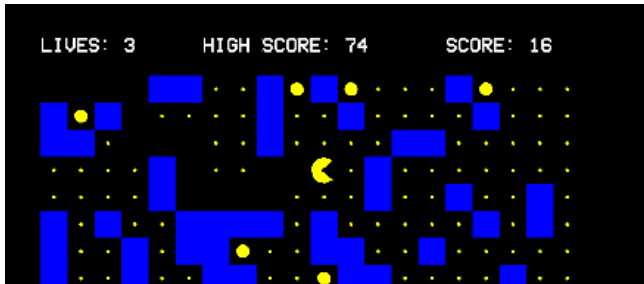

Test Case:	TC-015	Nome:	Perdita partita.
Riferimento:	REQ-07		
Descrizione:	Ogni volta che Pac-Man viene toccato perde una vita, quando perde tutte le sue vite il gioco finisce.		
Procedura:	<ol style="list-style-type: none"> 1. Fare partire L'applicativo in Unity. 2. Avviare una partita. 3. Usare i diversi tasti seguenti per fare muovere Pac-Man, in diverse direzioni, di un blocco sulla mappa: <ul style="list-style-type: none"> • Tasto "w" o freccia in su: sposta in su. • Tasto "a" o freccia a sinistra: sposta a sinistra. • Tasto "s" o freccia in giù: sposta in giù. • Tasto "d" o freccia a destra: sposta a destra. 4. Farsi prendere dal fantasma fino a quando non si ha più vite. 		
Risultati attesi:	Il gioco finisce e appare un bottone per ricominciare la partita oppure uscire dal gioco.		



Test Case:	TC-016	Nome:	Pausa.
Riferimento:	REQ-07		
Descrizione:	Cliccando un bottone si mette in pausa il gioco.		
Procedura:	<ol style="list-style-type: none"> 1. Fare partire L'applicativo in Unity. 2. Avviare una partita. 3. Cliccare il tasto "esc". 4. Selezionare "Retry". 5. Cliccare il tasto "esc". 6. Selezionare "Exit". 		
Risultati attesi:	In partita premendo "esc" si mette in pausa il gioco, poi premendo su "Retry" riinizia la partita e premendo su "Exit" si trona al menu.		

Test Case:	TC-017	Nome:	Controller.
Riferimento:	REQ-10		
Descrizione:	Si può collegare un gaming controller per giocare.		
Procedura:	<ol style="list-style-type: none"> 1. Fare partire L'applicativo in Unity. 2. Collegare il controller al dispositivo usato per giocare. 3. Navigare per il menu con il controller usando i diversi simboli (Quadrato, X e Cerchio). 4. Premere "X" per cominciare una partita. 5. Usare le frecce del joystick per muoversi 6. Quando si clicca su "Options" sul joystick viene messo in pausa. 		
Risultati attesi:	Si può navigare il menu, avviare la partita e giocare usando il controller.		

5.2 Risultati test

ID	Risultato	Commento	Data
TC-001	Passato	<p>La mappa viene generata correttamente.</p>  <p>Figura 41 - Mappa generata</p>	09.12.2022
TC-002	Passato	Pac-Man si muove in tutte e quattro le direzioni alla premuta dei tasti.	09.12.2022
TC-003	Passato	Se Pac-Man va contro un muro in qualsiasi direzione non gli viene permesso di andare avanti e rimane fermo.	09.12.2022
TC-004	Passato	Viene piazzata una pillola su ogni cella della mappa che non è un muro od una super-pillola.	09.12.2022
TC-005	Passato	<p>Una volta avviata la partita quando Pac-Man sala su una pillola viene mangiata e non compare più sulla mappa ed il punteggio aumenta.</p>  <p>Figura 42 - Manga pillole</p>	09.12.2022
TC-006	Passato	Il fantasma si avvicina a Pac-Man ogni tot secondi definiti da una variabile quando il giocatore rimane fermo.	09.12.2022
TC-007	Passato	Il fantasma si avvicina a Pac-Man ogni tot secondi definiti da una variabile e quando il giocatore si muove il fantasma lo segue e gli va sempre contro.	09.12.2022

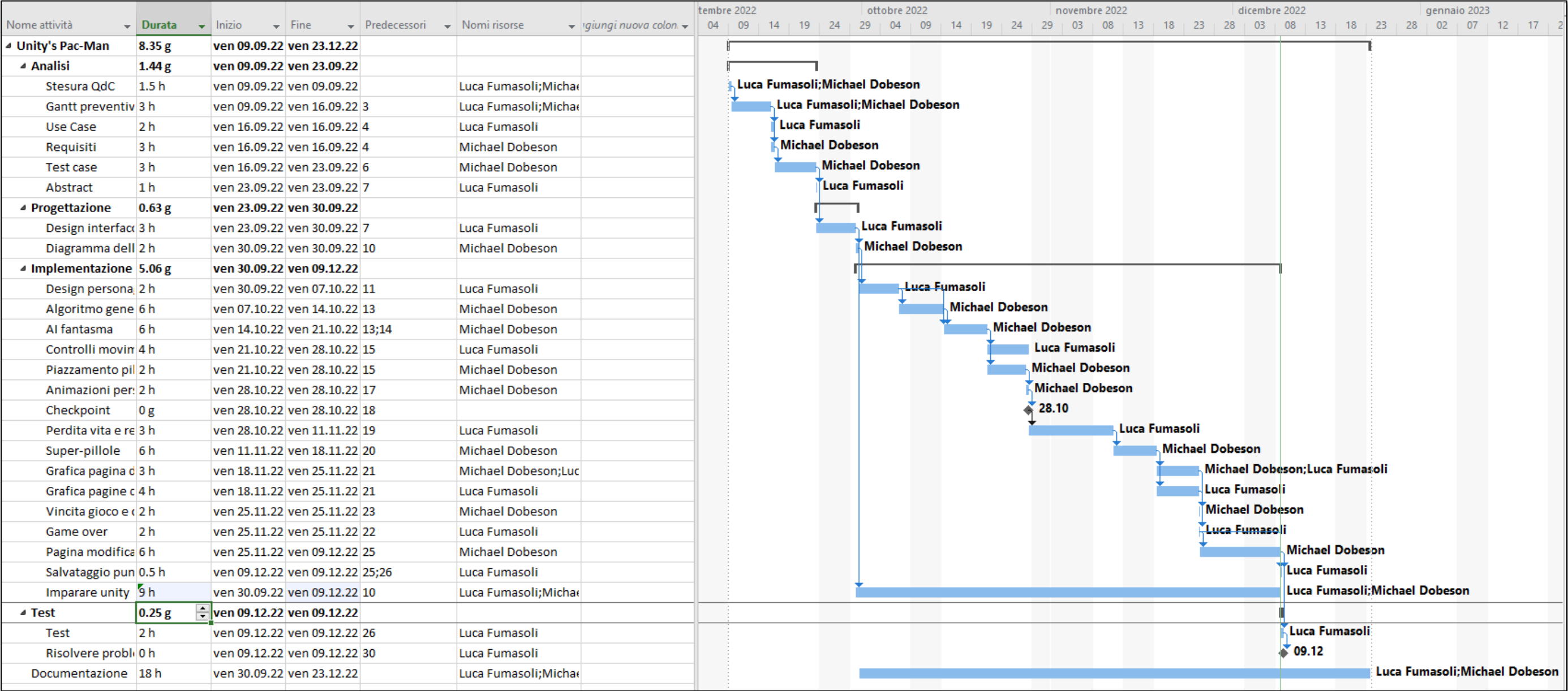
TC-008	Passato	Il fantasma si avvicina a Pac-Man ogni tot secondi definiti da una variabile attraverso il labirinto senza andare mai a sbattere od attraversare le mura.	09.12.2022
TC-009	Passato	Una volta che viene mangiata una Super-Pillola Pac-Man diventa invincibile.	09.12.2022
TC-010	Passato	Quando Pac-Man mangia una Super-Pillola il fantasma cambia animazione ed inizia a scappare dal giocatore. 	09.12.2022
		Figura 43 - Fantasma vittimizzato	
TC-011	Passato	Il punteggio incrementa di 1 ogni volta che una pillola viene mangiata. 	09.12.2022
		Figura 44 - Incremento punteggio	
TC-012	Passato	Se si ottiene un punteggio più alto dell'highscore questo viene salvato come nuovo highscore e viene mostrato in nuove partite anche se si riavvia l'applicazione.	09.12.2022
TC-013	Passato	Una volta che vengono mangiate tutte le pillole viene generata una nuova mappa dove si tiene il punteggio di prime e le vite aumentano della quantità settata nelle variabili. 	09.12.2022
		Figura 45 - Generazione nuova mappa	
TC-014	Passato	Quando si viene preso dal fantasma viene rimossa una vita ed il giocatore riparte dalla cella iniziale.	09.12.2022

TC-015	Passato	<p>Quando si viene presi dal fantasma e si hanno finito le vite la partita finisce ed appare la schermata di game over con il proprio punteggio e l'highscore.</p>  <p>Figura 46 - Game Over</p>	09.12.2022
TC-016	Passato	<p>Quando si preme esc appare la schermata di pausa e premendo su resume la partita continua mentre se si preme su main menu porta al menu principale correttamente.</p>  <p>Figura 47 - Pausa</p>	09.12.2022
TC-017	Passato	<p>Quando si premono i diversi tasti del joystick (Quadrato, X e Cerchio) si riesce a spostarsi nel menu. Quando si preme "X" parte una partita. Mentre si è nella partita si può usare le freccette per muovere il Pac-Man e quando si clicca "Options" sul joystick, la partita si mette in pausa.</p>	16.12.2022

5.3 Mancanze/limitazioni conosciute

Ci siamo pianificati in modo esatto al risultato finale secondo le nostre competenze e perciò non ci sono state mancanze né limitazioni nello svolgimento del progetto.

6
Consuntivo



7 Conclusioni

7.1 Sviluppi futuri

Come sviluppi futuri si potrebbero fare varie cose:

- Aggiungere più fantasmi e rendere il numero di fantasmi una variabile modificabile dal giocatore.
- Aggiungere una modalità classica in cui si gioca con la mappa del gioco originale.
- Aggiungere la modalità di gioco 3D.
- Multigiocatore, in cui un giocatore controlla *Pac-Man* ed un altro controlla il fantasma.
- La musica ed i suoni del gioco ed ambiente.

7.2 Considerazioni personali

7.2.1 Michael Dobeson

Il progetto in sé era una sfida essendo che le mie conoscenze di *Unity* erano molto limitate, però nel percorso del tempo sono migliorato e mi sono abituato e direi anche divertito. Per molti questo progetto non sembrare di essere qualcosa di importante nella vita o che cambierà il mondo, ma io credo che ogni cosa e ogni persona deve iniziare da qualche parte. Quindi credo che questo mi ha aiutato a fare il mio primo passo nel mondo di sviluppo.

7.2.2 Luca Fumasoli

Durante questo progetto ho imparato ad usare *Unity* per fare un videogioco, cosa che mi ha interessato imparare. Ho rispettato i tempi decisi all'inizio del progetto e siamo riusciti a finire il gioco. Sono riuscito a risolvere la maggior parte dei problemi che ho riscontrato durante il progetto abbastanza facilmente con qualche ricerca su internet.

8 Glossario

Termine	Descrizione
Axis	O asse in <i>Unity</i> sono dei diversi valori predefiniti che possono essere cambiati e abbinati a dei tasti.
Build	È una compilazione di tutto il progetto che poi consegna un'eseguibile per poi potere accedere all'applicativo.
Coroutine	È una funzionalità aggiuntiva negli <i>script C#</i> in <i>Unity</i> in cui puoi iniziare un metodo che funziona allo stesso tempo che fa un'altra azione.
Float	Un <i>float</i> è una variabile di tipo numerico qualsiasi.
FPS	<i>Frames per Second</i>
HideInInspector	È un comando che si mette (tra parentesi quadre) prima di una variabile pubblica in <i>Unity</i> per specificare che non si vuole mostrarlo nell'interfaccia di <i>Unity</i> .
IA	Intelligenza Artificiale o AI in inglese
Int	Un intero (o <i>int</i>) è un variabile di tipo numerico senza la virgola.
Joystick	Un <i>controller</i> per giocare, che nel mio caso ho usato un <i>controller PS4</i> .
Listener	È una proprietà, che dal nome, ascolta per un'azione dall'utente.
PlayerPrefs	È un valore che viene salvato dentro il file di <i>Unity</i> del gioco così anche alla chiusura dell'applicazione quei valori rimangono.
String	Una stringa (o <i>string</i>) è una variabile di tipo testuale.
Timeline	È il termine usato per specificare i tempi diversi di un'animazione.

9 Bibliografia

9.1 Sitografia

<https://www.codespeedy.com/detect-which-key-is-pressed-in-unity-c-sharp/>, Detect Which Key is Pressed in Unity C#, 28-10-2022

<https://answers.unity.com/questions/225213/c-countdown-timer.html>, Countdown Timer, 28-10-2022

<https://ezgif.com/split>, Split, 28-10-2022

<https://stackoverflow.com/questions/37341408/access-variables-functions-from-another-component>, Access Variables Functions From Another Component, 11-11-2022

<https://stackoverflow.com/questions/32332316/unity-how-can-i-show-hide-3d-text>, How Can I Show Hide 3D Text, 11-11-2022

<https://docs.unity3d.com/530/Documentation/ScriptReference/UI.Button-onClick.html>, UI Button OnClick, 18-11-2022

<https://stackoverflow.com/questions/62092866/c-sharp-unity-engine-how-to-limit-ui-slider>, C# Unity Engine How To Limit UI Slider, 25-11-2022