**Game Documentation**

****

**Project Name:**

**Team Name:**

Zombie Stratego

Zombie Team

**Team Members:**

Picatureanu Nicusor – junior programmer

Popescu Sabin – junior programmer



**Game Design Document Version Control**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Version** |  |  | **Date** |  |  | **Author** |  |  | **Change Description** |  |
|  |  |  |  |  |  |  |  |
| 1.0 | |  |  | Feb/09 | |  | Popescu Sabin | |  | Document created. | |
|  | |  |  |  | |  |  | |  |  | |
| 1.1 | |  |  | Feb/13 | |  | Picatureanu Nicusor | |  | Document updated. | |
|  | |  |  | | |  | | |  | | |
| 1.2 | |  | Feb/22 | | | Popescu Sabin | | | Game description and Functional | | |
|  |  |  |  |  |  |  |  |  | Specifications updated. | | |
|  |  |  |  |  |  |  |  |  | Sound & Music/Story bible added. | | |
| 1.3 | |  | Feb/23 | | | Popescu Sabin | | | Sound & Music assets added. | | |
| 1.4 | |  | Mar/19 | | | Picatureanu Nicusor | | | General document updated. | | |
| 1.5 | |  | Mar/22 | | | Picatureanu Nicusor | | | Functional Specifications and Art and | | |
|  |  |  |  |  |  |  |  |  | Video sections updated. | | |
| 1.6 | |  | Mar/29 | | | Picatureanu Nicusor | | | Level Requirements updated. | | |
| 1.7 | |  | Mar/30 | | | Popescu Sabin | | | Production Schedule updated. | | |
|  | |  |  | | |  | | |  | | |
| 1.8 | |  | Apr/08 | | | Popescu Sabin | | | Technical Specifications updated. | | |
|  |  |  |  |  |  |  | | |  |  |  |
| 1.9 | |  | Apr/10 | | | Popescu Sabin | | | Assets Pipeline added. | | |
| 2.0 | |  | Apr/12 | | | Picatureanu Nicusor | | | Technical Specifications updated. | | |

**Game Description**

Zombie Stratego este un Unity 3D game(joc), care pune la incercare gandirea logica si strategia pe care fiecare jucator trebuie sa o integreze si sa o puna in aplicare pentru a putea castiga . Ca si complexitate, jocul are o complexitate destul de mare datorita algoritmului folosit pentru AI acestui joc in modul Single Player . Asa cum s-a mentionat mai sus jocul are doua moduri de joc : Single player mode si Multiplayer mode . Prin multiplayer , face referire la 2 jucatori care se pot juca in LAN cu ajutorul modulelor din Unity.

***Design Goals***

Jocul urmareste atingerea urmatoarelor obiective:

1. Dezvoltarea gandirii logice a jucatorilor pasionati de jocurile de strategie
2. Punerea in scena a unui board game cu o grafica placuta , cu o scena unde se desfasoara actiunea , asezata si complexa.
3. Interactiunea cu alte persoane, prin intermediul modului Muliplayer
4. Dezvolatrea unui joc amuzant care se bazeaza pe:
   1. Descoperire
   2. Amuzament
   3. Putere
5. Arta si desing-ul care sunt credincioase stilului din dintr-ul atl joc numit PlantsVsZombies

***Influences & Sources***

Am cercetat in amanunt tot ce tine de jocul din care ne-am inspirat personajele noastre , si anume Plants Vs Zombies . Am plecat de la cei care au venit cu idea jocului , cum au ajuns la aceasta decizie si am ajuns la cei care au realizat codul citind documentatie despre fiecare . O atentie deosebita am acordat-o companiei care a scos pe piata acest joc pentru ca majoritatea jocurilor lor sunt de acest gen si asta ne-a ajutat pe noi la stabilirea mai multe reguli .

In plus echipa a cercetat mai multe benzi desenate pentru a putea stabili cum vor arata personajele noastre si toata grafica jocului .

Un rol foarte important l-au avut si jocurile studiate de noi , in special jocurile de acelasi gen pentru a putea aduce ceva inovativ care nu a mai fost implementat anterior.

***Target Market***

Utilizatorul ideal al aplicatiei :

1. Persoana (nu conteaza genul sau varsta)
2. Are un Pc pe care se poate juca
3. Este un gamer;
4. Doritor de dezvoltarea gandirii si mai ales de interactiunea cu alte persoane care pot cu care sa isi incerce strategia gandita.

*Inteligenta artificiala in industria jocurilor*

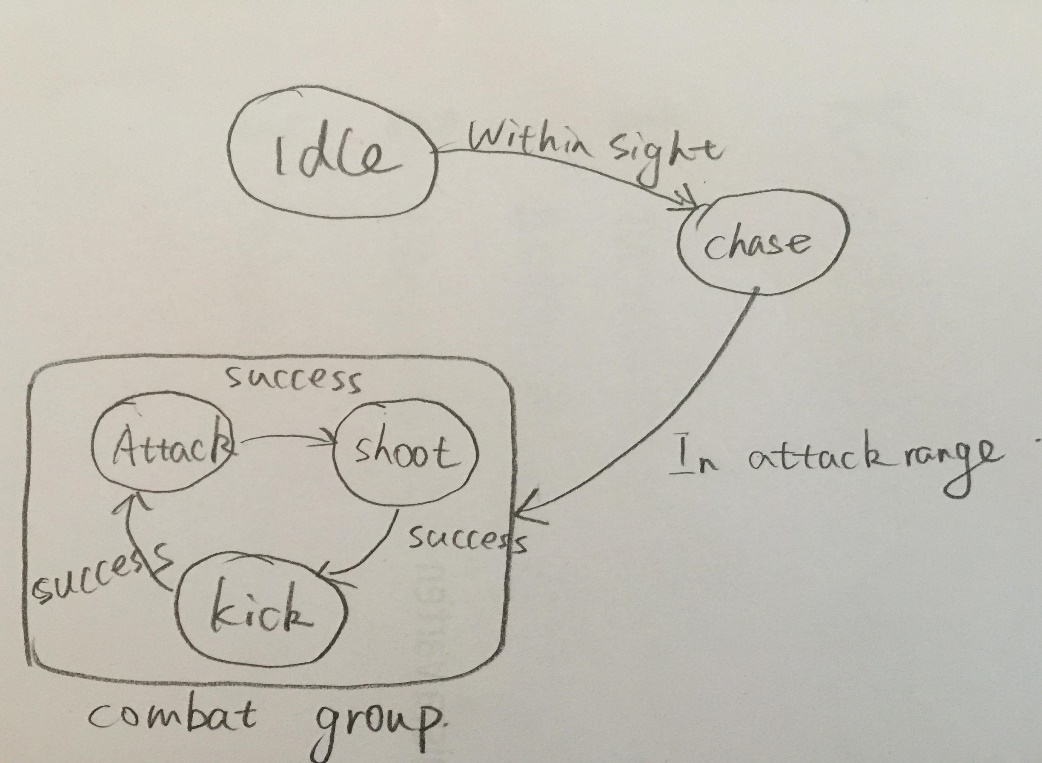
*Introduction*

Acest capitol va explica modul în care AI a fost utilizat în industria jocurilor. De asemenea, acest capitol va explica două modalități diferite de a realiza designul AI modern și, de asemenea, istoricul lung al progresului utilizării AI în industria jocurilor. Scopul acestui capitol este de a da o scurtă explicație cu privire la teoria din spatele designului modern AI.  
  
AI a fost folosit în industria jocurilor din anii 1970 (Wikipedia, citat 01.01.2017). Un joc bun poate fi uneori stabilit de un AI bun. În prezent, AI are un impact tot mai mare nu numai în jocurile video, ci și în industria jocurilor mobile cu dezvoltare rapidă.

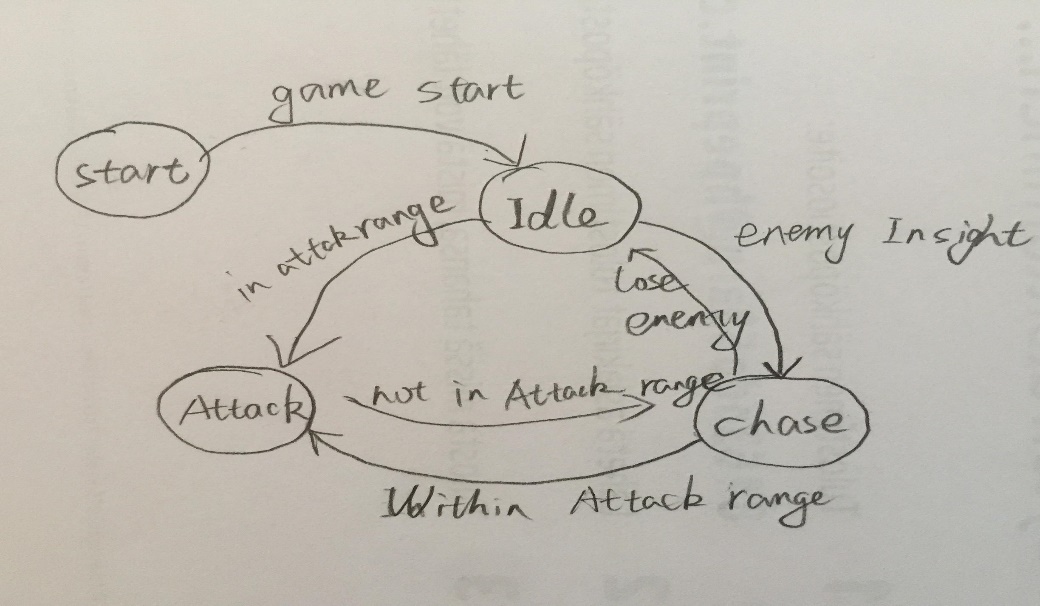
Progresul Inteligentei artificiale in industria jocurilor

În stadiul timpuriu al industriei de jocuri de noroc, AI a constat în mare parte din câteva reguli simple. Nu a fost încă partea principală a dezvoltării jocurilor (Wikipedia, citat 02.01.2017). Primul joc care a introdus publicul un AI cu adevărat avansat este un joc numit Half Life. Inamicul AI al acestui joc folosește o mașină de stat pe bază de program, care face ca inamicul să prezinte un comportament diferit în anumite situații critice (Woodcock, S. 1998, p.1).  
  
Mașina de stat este utilizată pe scară largă în industria jocurilor pentru a realiza un comportament AI comun, dar există unele probleme semnificative care apar cu state machine-ul. Mai întâi de toate, deși state machine-ul este ușor de proiectat, a devenit foarte greu să se mențină atunci când se adaugă mai multă stare, pentru că de fiecare dată când a fost adăugat un nou stat, trebuiau să fie luate în considerare toate tranzițiile legate de această stare și alt stat. În prezent, dezvoltatorii au găsit o nouă modalitate de a implementa acele comportamente. Se numește arborele de comportament. Are o logică încapsulată într-o structură ierarhică, de atunci un arbore de comportament a fost o formă standard a industriei. În comparație cu mașina de stat, aceasta oferă o soluție scalabilă pentru adăugarea logicii la AI. Este cu siguranță o soluție mai bună pentru un proiect mare (Pereiro, R 2015, p.1)

AI are o istorie relativ lungă și sa schimbat în ultimii ani. Cea mai cunoscută descoperire a fost atunci când Google alpha a bătut cel mai bun jucător Go din lume. Aceasta include tehnicile de învățare a mașinilor și de căutare a copacilor (Reese, H. Cited 12.14.2016). AI este întotdeauna asociată cu informatica, dar algoritmii utilizați în AI au venit din multe alte domenii, cum ar fi matematica și fizica. Scopul AI este de a crea un agent mai uman ca să ajute oamenii să rezolve diferite sarcini (Wikipedia, citat 01.03.2017).



*FIGURE 1. State machine example*



*FIGURE 2. Hierarchical state machine example*

State machine-ul este un model folosit pentru a controla fluxul de execuție, care are nevoie doar de două componente simple, o stare și o tranziție.  
Statul reprezintă fluxul curent executat. Tranziția este folosită pentru a traduce într-o altă stare atunci când îndeplinește condiția. Machine state-ul este ca și creierul inamicului, fiecare stat este ca o sarcină pentru ca inamicul să termine. În figura 1, imaginea prezintă un exemplu comun al mașinii de stat. Există patru stări: pornire, inactivitate, atac și urmărire, fiecare dintre ele are o tranziție spre alt stat.  
State machine-ul are o versiune imbunatatita care a numit automat finit ierarhic (Jorge, P. 2016, p.98), este asemanatoare cu automatul finit. Diferența este că dezvoltatorul poate face un grup de anumite state, iar toți membrii grupului pot împărți aceeași tranziție către alte grupuri, ceea ce va oferi o mai mare reutilizare. În figura 2, imaginea arată un exemplu de mașină de stare ierarhică. Există trei state într-un grup de luptă, toți împărtășesc aceeași tranziție de la o stare de urmărire. Este ca un sistem de buclă în interiorul grupului de luptă în timp ce starea reușește.

Prezentul joc are un AI destul de complex deoarece acesta invata singur din propriile greseli/mutari devenind din ce in ce mai inteligent odata cu trecerea timpului in joc . Acesta se bazeaza pe urmatorul algoritm (bazat pe metodele):

int \_Evaluate()

{

float pieceDifference = 0;

float whiteWeight = 0;

float blackWeight = 0;

foreach(Tile tile in \_whitePieces)

{

whiteWeight += \_weight.GetBoardWeight(tile.CurrentPiece.Type, tile.CurrentPiece.position, Piece.playerColor.WHITE);

}

foreach (Tile tile in \_blackPieces)

{

blackWeight += \_weight.GetBoardWeight(tile.CurrentPiece.Type, tile.CurrentPiece.position, Piece.playerColor.BLACK);

}

pieceDifference = (\_blackScore + (blackWeight / 100)) - (\_whiteScore + (whiteWeight / 100));

return Mathf.RoundToInt(pieceDifference \* 100);

} – aceasta metoda calculeaza cea mai buna pozitie pentru a muta piesa

Move \_CreateMove(Tile tile, Tile move)

{

Move tempMove = new Move();

tempMove.firstPosition = tile;

tempMove.pieceMoved = tile.CurrentPiece;

tempMove.secondPosition = move;

if (move.CurrentPiece != null)

{

tempMove.pieceKilled = move.CurrentPiece;

}

return tempMove;

} – metoda care realizeza miscarea propriu - zisa

Si doua metode care au la baza AI-ul: void \_UndoFakeMove()

{

Move tempMove = moveStack.Pop();

Tile movedTo = tempMove.secondPosition;

Tile movedFrom = tempMove.firstPosition;

Piece pieceKilled = tempMove.pieceKilled;

Piece pieceMoved = tempMove.pieceMoved;

movedFrom.CurrentPiece = movedTo.CurrentPiece;

if (pieceKilled != null)

{

movedTo.CurrentPiece = pieceKilled;

}

else

{

movedTo.CurrentPiece = null;

}

}

void \_DoFakeMove(Tile currentTile, Tile targetTile)

{

targetTile.SwapFakePieces(currentTile.CurrentPiece);

currentTile.CurrentPiece = null;

}

List<Move> \_GetMoves(Piece.playerColor color)

{

List<Move> turnMove = new List<Move>();

List<Tile> pieces = new List<Tile>();

if (color == Piece.playerColor.BLACK)

pieces = \_blackPieces;

else pieces = \_whitePieces;

foreach (Tile tile in pieces)

{

MoveFactory factory = new MoveFactory(\_board);

List<Move> pieceMoves = factory.GetMoves(tile.CurrentPiece, tile.Position);

foreach(Move move in pieceMoves)

{

Move newMove = \_CreateMove(move.firstPosition, move.secondPosition);

turnMove.Add(newMove);

}

}

return turnMove;

}

Aceste doua metode sunt cele care fac ca algoritmul sa fie unul complex pentru ca in momentul in care o piesa o sa fie distrusa sau o sa fie slabit zidul care protejeaza steguletul atunci algoritmul o sa incerce sa resolve problema si mai ales sa imbanateasca miscarile .

Pentru realizarea acestui algoritm , Unity pune la dispozitie un editor in care sa fie scrise aceste metode , importand modulele specifice de Unity :

using UnityEngine;

using System.Collections;

using System.Collections.Generic;

. Depinde de preferintele fiecarui programator , script-ul poate fi creat utilizand orice alt IDE pentu limbajul de programare C#.

AI din joc este capabil sa aranjeze piese/ pionii astfel incat sa isi surprinda adversarul si anume, cum prea bine se stie , fiecare pion are o putere anume iar algoritmul isi va alege pioni cei mai puternici si ii va aseza in linia de la mijloc jumatate din acestia si cealalta jumatate o sa fie distribuita in mod egal atat in prima linie de pioni cat si in ultima linie de pioni .



**Functional Specifications**

Mecanicile jocului

***Core Game Play***

|  |  |  |
| --- | --- | --- |
| **purpose** | | User should be able to see on the the monitor a game board consisting of his/her pieces and his/her opponent's pieces |
| **inputs** | | Input will be data from a Board class that contains an array of Terrain class objects. The Terrain class will contain the data of how a square should be displayed graphicly.   Additional input will come from a Player class that contains a list of active pieces. Each piece will be an instance of a Piece class that contains the piece's current position on the board. |
| **processing** | | The board and all player pieces as well as opponent pieces are displayed to the user |
| **outputs** | | return true if no errors displaying graphics, else return false |
|  | |  |
| **purpose** | | User should be able to move pieces by clicking the mouse over a piece and dragging it to a new location | | |
| **inputs** | | Input will be from the mouse when a user clicks. The mouse (x,y) position will be passed and also the board information will be inputted | | |
| **processing** | | Checks whether there is a player piece in the location of the mouse click. Checks whether the location where the mouse is released is a valid place on the board, if not the piece will return to it's original location  Updates the board with the new piece location  Updates the player piece with the new location | | |
| **outputs** | | returns true if the piece was moved false otherwise | | |

|  |  |
| --- | --- |
| **purpose** | User should be able to connect to another player to play the game. |
| **inputs** | A TCP/IP connection with the other user, the other user's piece positions |
| **processing** | Updates the map board with the other player's piece locations. |
| **outputs** | none |

|  |  |
| --- | --- |
| **purpose** | User should be able to attack/capture another player's piece using a valid chess move, or special attack |
| **inputs** | The piece that is attacking, the piece that is being attacked |
| **processing** | checks whether the attacking piece is using a valid chess move or special ability. Checks whether the piece being attacked is an opponent piece. Deducts the hit points of the piece being attacked if the move is valid |
| **outputs** | true if attack is valid false otherwise |

|  |  |
| --- | --- |
| **purpose** | User is able to win the game by checkmating king |
| **inputs** | The current game board |
| **processing** | Checks opposing piece's king to see if it is in checkmate |
| **outputs** | true if opposing king is in checkmate false otherwise |

|  |  |
| --- | --- |
| **purpose** | User gets points for winning match based on a point differential. (i.e. if an underdog defeats a power player, they receive more points). |
| **inputs** | the player who won the previous match, the point differential of the players |
| **processing** | Uses a formula to determine the points to award another player for winning a match |
| **outputs** | the points that are to be awarded to the winning player |

|  |  |
| --- | --- |
| **purpose** | User can surrender |
| **inputs** | The player that is surrendering |
| **processing** | Surrenders the match and awards the opposing player as the winner of the match. |
| **outputs** | none |

|  |  |
| --- | --- |
| **purpose** | User can upgrade a piece by using points |
| **inputs** | The piece that is to be upgraded, the current value of the players points |
| **processing** | Checks if the player has enough points to upgrade the piece and adds the upgrade to the piece if so. Deducts the points from the users total points. |
| **outputs** | returns true if the piece was upgraded false otherwise |

|  |  |
| --- | --- |
| **purpose** | User can buy new pieces (i.e. additional queens etc) |
| **inputs** | The piece to be bought, the players current point total |
| **processing** | Checks if the player has enough points to purchase a piece if they do then the piece is added to the players list of pieces. Deducts the points from the users point total |
| **outputs** | returns true if the piece was added false otherwise |

***Game Flow***

Actiuni pe care jucatorul le poate realiaza:

* Mutarea pionilor in stanga/ dreapte/ inainte/inapoi
* Cumpararea pionilor in timpul jocului
* Schimbarea strategiei
* Aplicarea deciziei de a ataca adversarul

***Game Play Elements***

Urmatoarea enumeratie reprezinta elementele cu care o sa interactioneze jucatorul in timpul jocului:

* Introducerea numelui . Default numele are o valoare random
* Alegerea modului de joc : Single Player sau MultiPlayer
* Alegerea strategiei
* Mutarea pionilor
* Eliminare pionilor
* In coltul din dreapta sus se gaseste bara cu banii ramasi
* Compararea unor noi Pioni
* Punerea pe pauza a jocului

In cazul in care este selectat modul Multiplayer atunci utilizatorul:

* Va avea la dispozitie creearea unui noi sesiuni(room)
* A la join la una din sesiunile existente

***Game Physics and Statistics***

Fizica in joc functioneaza ca atare:

* Jucatorii se deplaseaza intr-un plan de derulare 3D, fie de la stanga la dreapta fie de la dreapts stanga
* Jucatorii pot traversa traseele cu caractere inamice daca nu sunt lovite de un atac , cee ace il impinge inapoi..
* Pioni sunt distrusi in momentul in care sunt loviti dar in spatele acestora pot exista diferite arme ascunse precum bombe.
* Cand un obstacol se afla in fara caracterului care poate fi redat , nu se poate continua in aceasta directie
* Trebuie sa existe o gravitate

***Multiplayer***

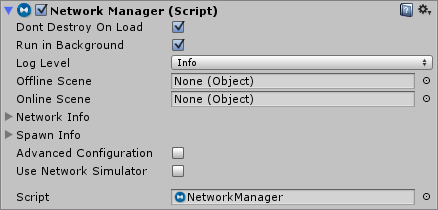
Ceea ce am folosit pentru realizarea modului multiplayer se numeste Unity Network care pune la dispozitie mai multe metode care o sa fie descrise in ceea ce urmeaza. In cee ace priveste ceea ce e nevoie in proiect , acestea sunt :

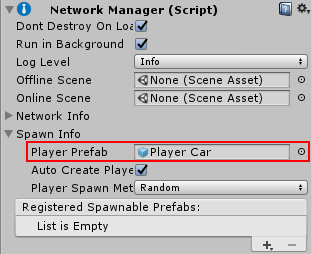
* Un manager de retea
* O interfata cu utilizatorul
* Network prefabs player
* Scripturi si GamObjects care sunt multiplayer-constienti

Exista , de asemenea , cateva concept importante pe care trebuie sa le intelegeti sis a faceti alegeri atunci cand va construiti jocul . Aceste concepte pot fi rezumate pe larg ca:

* Relatia dintre un client , un sever si o gazda
* Ideea de autoritate asupra GameObjects si actiuni

Managerul de retea este responsabil pentru gestionarea retelei aspecte ale jocului. Trebuie sa existe si un manager de retea activ in scena la un moment dat pentru ca totul sa functioneze asa cum trebuie.





Algoritmul pentru multiplayer se bazeaza pe faptul in scena trebuie sa exista un manager de retea care sa permita creeare unei sesiunii . Acestui manager de retea i-a fost asignat un script in care sunt folosite metodele pentru creearea unei sesiuni intre doi jucatori care sunt conectati la aceeasi retea . In cazul in care un al treilea jucator incerca creearea unei sesiuni aceasta va fi creeata dar va fi pusa intr-o coada unde va astepta pana cand un jucator din sesiunea deja existent sa fie liber .

public void OnGameStartedOccured(string \_Sender, string \_RoomId, string \_NextTurn) {

Debug.Log("OnGameStartedOccured: " + \_Sender + " " + \_RoomId + " " + \_NextTurn);

if(\_Sender != username) {

realEnemyUsername = ParseUsername(\_Sender);

}

currentUsernameTurn = \_NextTurn;

ParseUsername(\_NextTurn);

GameManager.CURRENT\_TURN = GameSide.LeftSide;

if(currentUsernameTurn == username) {

playerSide = GameSide.LeftSide;

isMyTurn = true;

var awaySoldiers = MiniJSON.Json.Deserialize(data["AwayPlayer"].ToString()) as List<object>;

SoldierManager.Instance.InitEnemyBoard(awaySoldiers);

}

else {

playerSide = GameSide.RightSide;

isMyTurn = false;

var homeSoldiers = MiniJSON.Json.Deserialize(data["HomePlayer" + currentUsernameTurn].ToString()) as List<object>;

SoldierManager.Instance.InitEnemyBoard(homeSoldiers);

SoldierManager.Instance.FlipSide();

}

SoldierManager.Instance.HideAllSoldiers();

SwitchToGameScene();

}

In momentul in care sesiunea este valida si cei doi jucatori accepta participarea atunci se va initializa sesiunea prin cel de-al doilea script care este atribut managerului de network si anume:

private void Init() {

listener = new Listener();

rooms = new List<string>();

data = new Dictionary<string, object> {

{ "Password", "12345" }

};

WarpClient.initialize(Globals.API\_KEY, Globals.SECRET\_KEY);

WarpClient.GetInstance().AddConnectionRequestListener(listener);

WarpClient.GetInstance().AddChatRequestListener(listener);

WarpClient.GetInstance().AddUpdateRequestListener(listener);

WarpClient.GetInstance().AddLobbyRequestListener(listener);

WarpClient.GetInstance().AddNotificationListener(listener);

WarpClient.GetInstance().AddRoomRequestListener(listener);

WarpClient.GetInstance().AddZoneRequestListener(listener);

WarpClient.GetInstance().AddTurnBasedRoomRequestListener(listener);

DontDestroyOnLoad(WarpClient.GetInstance());

username = System.DateTime.UtcNow.Ticks.ToString();

}

public void ConnectGame() {

GameView.SetText("StatusTxt", "Connecting...");

realUsername = PlayerPrefs.GetString("Username", "No-Name");

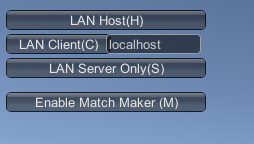
username = realUsername + "#" + username;

data["HomePlayer" + username] = MiniJSON.Json.Serialize(GetLocalSoldiers().ToArray());

WarpClient.GetInstance().Connect(username);

//WarpClient.GetInstance().GetRoomsInRange(1, 2);

} }

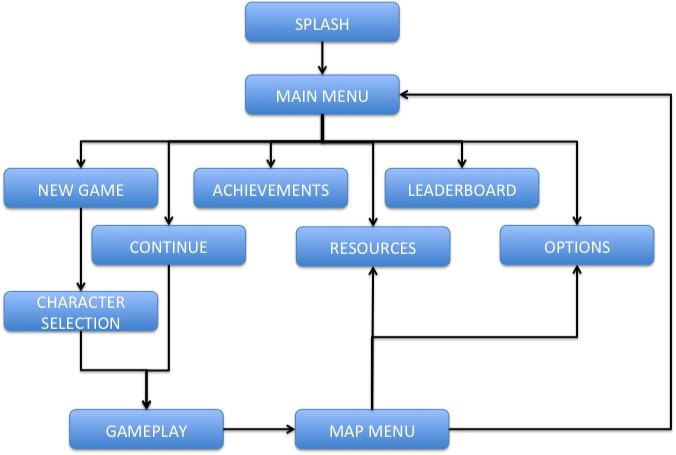


Dupa atribuirea celor 2 scripturi , in partea dreapta a editorul de Unity se poate selecta modul `LAN` prin `apasarea` casutei din dreptul acestuia si anume : `enable`.

Odata ce sesiune este valida cu 2 jucatori valizi atunci jocul va incepe cu un delay de maxim 3 secunde pana cand se va valida sesiunea .

**User Interface**

***Flowchart***



***Functional Requirements***

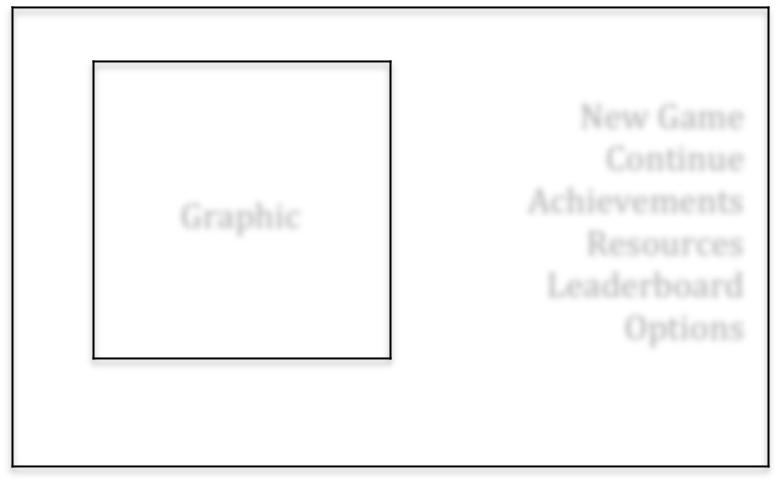
* Splash : acesta este ecranul initial si introdule logo-ul jocului
* Meniul principal: prezinta toate optiunile posibile pentru utilizator, precum si o imagine grafica captivanta care introduce aspectul si simtul jocului si ale seriei.
* Selectia modului de joc: Single player sau MultiPlayer
* Realizari: galerie in care utilizatorul poate verifica toate premiile pe care le-a obtinut precum si cele pe care le poate inca debloca.
* Optiuni: ofera utilizatorului posibilitatea dea controla unele dintre atributele jocului , cum ar fi volumul sonor si volumul de muzica

***Mockups***

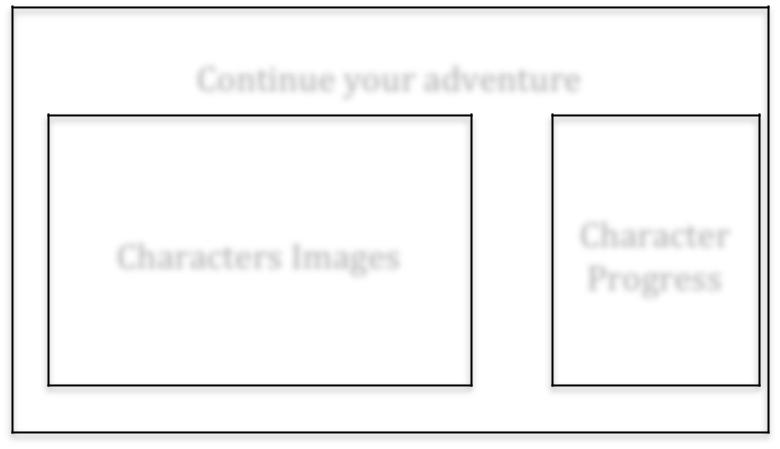
1. Splash Page

Zombie Stratego

2. Main Menu



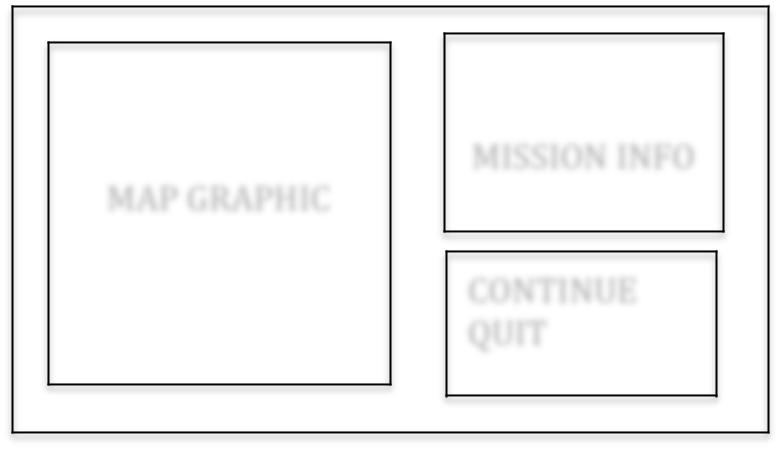
3. Continue



4. Options



5. Map Menu



6. Gameplay

Player2

Enemy/

Player1

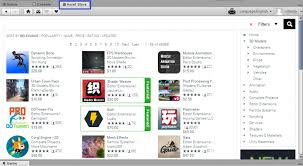
|  |  |
| --- | --- |
| Char | Health Bar |
| Portr |  |
| Power Bar |
| ait |
|  |  |

***GUI Objects***

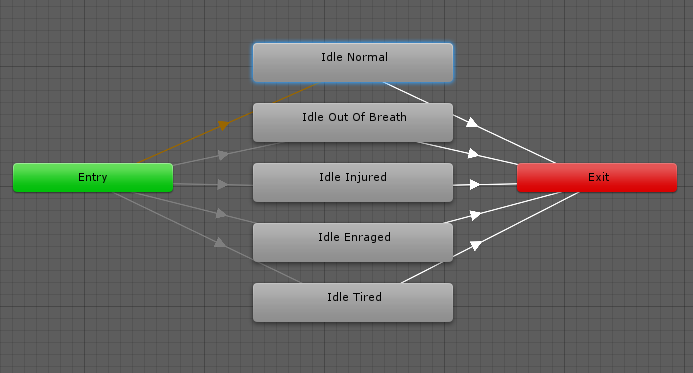


**Art and Video**

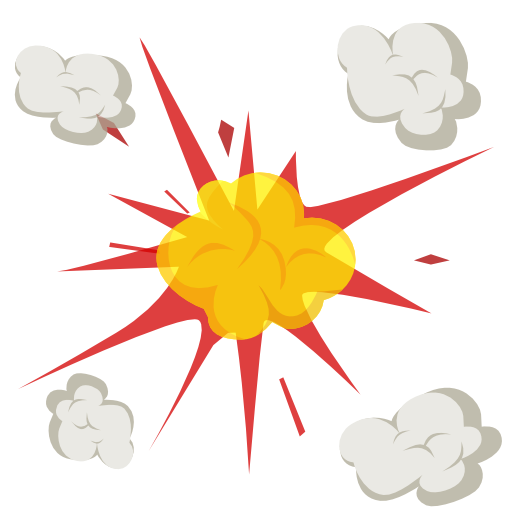
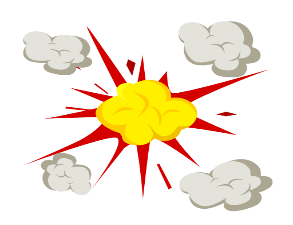
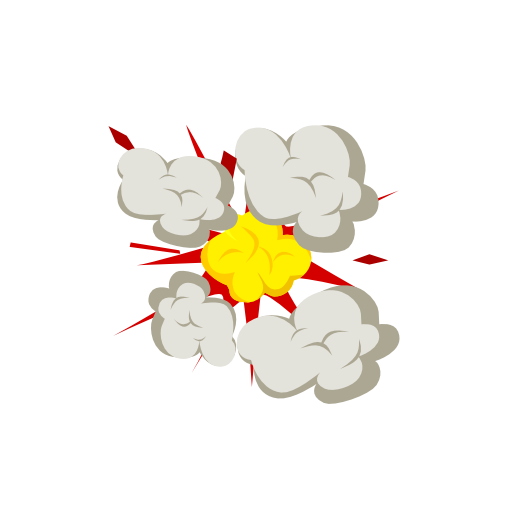
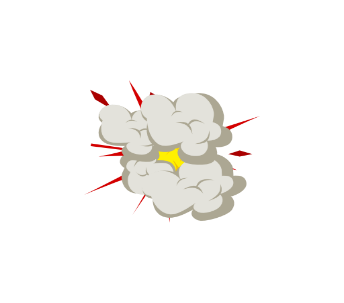
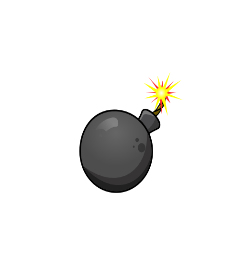
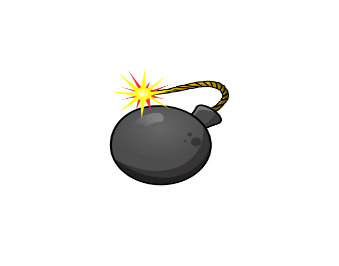
In ceaa ce priveste asseturile folosite pentru realizarea grafici , acestea au fost descarcate si integrate in joc folosing Asset Store , `magazinul` online al Unity-ului .



Pentru a integra asseturile in proiect este nevoie doar de descarcarea lor si de folosirea lor in proiect. Toate asset-urile folosite in acest joc au fost free , care aveau deja animatie , singulul lucru care mai ramane de facut este doar state machine-ul pentru fiecare element in parte :



In cee ace priveste toate efectele si materialele folosite , acestea sunt imagini integrate in state machines .

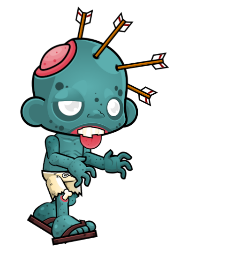


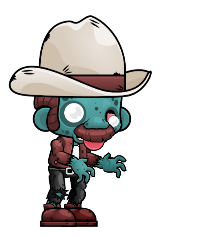
**Story**

***Player Characters***

Caracterele din acest joc sunt reprezentate de personajele din zombie vs Plants game care este un joc cu peste 50 de milioane de descarcari . Inamicii sunt aceleasi personaje .

Ce diferentiaza pionii intre ei este suma de bani care este platita pentru achizitionarea acestora si ce se afla in spatele lor dupa ce sunt atacati . Acest algoritm este unul random , astfel incat dintr-o lista de surprize este aleasa una singura sau chiar niciuna .



***Story theme***

Jucatorul care isi creeza propria tactica trebuie sa avanseze si sa distruga zidul din fata acestuia pentru a cucerii teritoriul advers .

***Story Outline***

In modul Online , cand 2 jucatori sunt in retea , povestea este aceeasi doar ca e mult mai distractiv si mai competitiv pentru ca altii jucatori pot fi mult mai buni decat AI-ul din modul single player .

**Technical Specifications**

***Game engine***

Am folosit un motor de joc pentru a ne dezvolta jocul numit Unity3D dezvoltat de Unity Technologies. Un motor de joc este un sistem conceput pentru a dezvolta jocuri pentru diverse platforme, cum ar fi console, computere și dispozitive portabile precum smartphone-urile.

***Platform and OS***

Unity oferă posibilitatea de a exporta jocul pe diverse platforme, printre care: iOS, Mac Standalone, Windows Standalone, Web, Nintendo Wii, Xbox 360 PS3 și Android.

Acest joc este dezvoltat special pentru PC.

***Code Objects***

**Player Object:**

Obiectul jucătorului este denumit după caracterul curent care este redat și poate fi identificat de eticheta "Player". Obiectul are scriptul Player.cs pe acesta, care este folosit pentru a acționa în funcție de intrarea utilizatorului, aceasta se face prin ascultarea evenimentelor de la ButtonManage.cs. Acest obiect este, de asemenea, conectat la extensia editorului ex2D, astfel încât playerul poate fi animat în funcție de acțiunile sale.

**Enemy Objects:**

Obiectele inamice vin în 3 variante diferite și sunt identificate de eticheta "Enemy". Tipul său este stabilit de o valoare booleană în scriptul Enemy.cs care este atașat obiectului inamic. Diferența dintre cele 3 tipuri de dușmani este cantitatea de experiență pe care o oferă utilizatorului după ce a fost omorât, cantitatea de sănătate pe care o au și cantitatea de daune pe care o suportă. Asta inseamna ca inamicul este cel mai slab si inamicul este cel mai puternic. Scriptul Enemy.cs oferă, de asemenea, inamicului un AI, care va fi explicat mai târziu.

**Camera Object:**

Camera Object este o cameră ortografică care există în fiecare scenă a jocului. Acest obiect a atasat 2 scripturi. Primul se numește rayCastScript.cs Acest script gestionează toate raze-urile difuzate când utilizatorul atinge ecranul. Scopul scripturilor este să lase scriptul ButtonManager.cs, care este de asemenea atașat la obiectul camerei, să știe dacă utilizatorul apasă sau nu un buton.

**Button Objects:**

Jocul are 6 obiecte în butonul de joc, acestea sunt: LeftButton, RightButton, Attack, Defend, Power, IngameMenu. Toate aceste obiecte sunt legate de scriptul buttonManager de pe obiectul camerei.

**GameManager:**

Acest obiect este numit GameManage și are atașat scriptul GameManager.cs. Obiectul gestionează lucruri precum inamicul și punctele pe care jucătorul este capabil să le cheltuiască pe statistici după nivelare. Se ocupă de dușmani, făcând o listă cu toți dușmanii din scena actuală. Apoi, obiectele Trigger pot cere managerului de joc un dușman pe listă atunci când declanșatorul trebuie să curețe unul.

***Design-ul architectural***

Designul arhitectural este format din toate clasele și relația dintre ele. Pentru a face lucrurile

mai vizibile fiecare clasă este afișată separat cu membrii de date și operations.

Toate GUI legate datele de membru nu sunt incluse.

Nu se afișează, de asemenea, toți agenții de manipulare, modele de tabelă și redare de tabele. Este afișată o diagramă de clasă completă, dar nu exclude informarea a prezentat în diagramele individuale de clasă.

Acest lucru a fost făcut în scopuri de vizibilitate.

Grupurile de pachete sunt prezentate și în diagrame separate fără relații, deoarece clasa completă

diagrama prezintă relațiile.

**Classes**

**class MenuController:**

private Slider moneySlider, musicSlider, sfxSlider;

private SoundManager soundManager;

private InputField usernameField;

private void Start()

public void PlayerSelectedInEditMode(SoldierBtn soldierSelected)

public void SaveChangesInEditMode()

public void StartSingleGame()

public void Multiplayer()

public void ChangeUsername()

Aceasta clasa este cea care controleaza intreaga functionalitate a meniului principal.

class GameController : MonoBehaviour

private Slider musicSlider, sfxSlider;

private SoundManager soundManager;

private void Start()

public void Options()

public void Back()

public void MoveSfxSlider()

public void MoveMusicSlider()

public void GamePause()

public void ResumeGame()

public void ResetGame()

public void QuitGame()

Aceasta este clasa care se ocupa de toate functionalitatea jocului . In momentul in care se incarca jocul aceasta clasa este asignata Object game-ului iar cand un utilizator doreste sa execute o anumita comanda , aceasta va fi rulata din acest clasa .

Urmatoarea clasa este cea care se ocupa de sunetul jocului :

public class SoundManager : Singleton<SoundManager> {

private AudioSource music;

private AudioSource sfx;

public AudioSource Music { get { return music; } }

public AudioSource SFX { get { return sfx; } }

[SerializeField] private AudioClip buttonPress;

[SerializeField] private AudioClip singlePlayerWin;

[SerializeField] private AudioClip singlePlayerLose;

[SerializeField] private AudioClip zombieBought;

[SerializeField] private AudioClip bombBought;

[SerializeField] private AudioClip flagBought;

[SerializeField] private AudioClip bombExplode;

[SerializeField] private AudioClip soldierSold;

[SerializeField] private AudioClip zombieWalk;

[SerializeField] private AudioClip zombieWalkShort;

[SerializeField] private AudioClip zombieDie;

[SerializeField] private AudioClip zombieAttack;

[SerializeField] private AudioClip zombieCancelBomb;

[SerializeField] private AudioClip clockTickOne;

[SerializeField] private AudioClip clockTickTwo;

[SerializeField] private AudioClip description;

[SerializeField] private AudioClip copycat;

[SerializeField] private AudioClip explode;

[SerializeField] private AudioClip inGameMusic;

public AudioClip ButtonPress { get { return buttonPress; } }

public AudioClip SinglePlayerWin { get { return singlePlayerWin; } }

public AudioClip SinglePlayerLose { get { return singlePlayerLose; } }

public AudioClip ZombieBought { get { return zombieBought; } }

public AudioClip BombBought { get { return bombBought; } }

public AudioClip FlagBought { get { return flagBought; } }

public AudioClip InGameMusic { get { return inGameMusic; } }

public AudioClip BombExplode { get { return bombExplode; } }

public AudioClip SoldierSold { get { return soldierSold; } }

public AudioClip ZombieWalk { get { return zombieWalk; } }

public AudioClip ZombieWalkShort { get { return zombieWalkShort; } }

public AudioClip ZombieDie { get { return zombieDie; } }

public AudioClip ZombieAttack { get { return zombieAttack; } }

public AudioClip ZombieCancelBomb { get { return zombieCancelBomb; } }

public AudioClip ClockTickOne { get { return clockTickOne; } }

public AudioClip ClockTickTwo { get { return clockTickTwo; } }

public AudioClip Description { get { return description; } }

public AudioClip Copycat { get { return copycat; } }

public AudioClip Explode { get { return explode; } }

private void Awake() {

DontDestroyOnLoad(this);

AudioSource[] audioSources = GetComponents<AudioSource>();

music = audioSources[0];

sfx = audioSources[1];

}

}

Clasa specifica fiecarui jucator controland functionalitatea fiecaruia :

public abstract class PlayerSoldier: MonoBehaviour {

[SerializeField] protected short rank;

[SerializeField] private int price;

[SerializeField] protected float offset\_x;

[SerializeField] protected float offset\_y;

[SerializeField] protected string soldierName;

[SerializeField] protected string description;

[SerializeField] protected string specialAbilityDescription;

private StrategyEditor strategyEditor;

protected SpriteRenderer spriteRenderer;

private Color blueColor = new Color(0, 0, 1, 0.6f);

private Color redColor = new Color(1, 0, 0, 0.6f);

private Color defaultColor = new Color(1, 1, 1, 1);

protected float originOffsetX, originOffsetY;

protected Vector3 originPosition;

protected Animator anim;

protected RuntimeAnimatorController originAnim;

protected PolygonCollider2D playerCollider;

protected bool isHidden;

protected Sprite sprite;

private float clickTime;

public short Rank { get { return rank; } set { rank = value; } }

public int Price { get { return price; } }

public float OffsetX { get { return offset\_x; } }

public float OffsetY { get { return offset\_y; } }

public Tile CurrentTile { get; set; }

public GameSide CurrentSide { get; set; }

public bool IsHidden { get { return isHidden; } }

public Animator Anim { get { return anim; } set { anim = value; } }

public RuntimeAnimatorController OriginAnim { get { return originAnim; } set { originAnim = value; } }

public PolygonCollider2D PlayerCollider { get { return playerCollider; } }

public Vector3 OriginPosition { get { return originPosition; } }

public Sprite Sprite { get { return sprite; } }

public string SoldierName { get { return soldierName; } }

public string Description { get { return description; } }

public string SpecialAbilityDescription { get { return specialAbilityDescription; } }

public abstract void SoldierPlacedInEditMode(bool isSoundActivated);

public abstract void MakeNoise();

public abstract void SetWarTag();

private void Awake() {

anim = GetComponent<Animator>();

originAnim = anim.runtimeAnimatorController;

playerCollider = GetComponent<PolygonCollider2D>();

spriteRenderer = GetComponent<SpriteRenderer>();

sprite = spriteRenderer.sprite;

strategyEditor = StrategyEditor.Instance;

originOffsetX = offset\_x;

originOffsetY = offset\_y; }}

Clasa care se va ocupa de functionalitatea de multiplayer este:

public class MultiPlayerManager: Singleton<MultiPlayerManager> {

private Dictionary<string, object> data;

private Listener listener;

private string username = string.Empty;

private string realUsername = string.Empty;

private string realEnemyUsername = string.Empty;

private List<string> rooms;

private int index = 0;

//

private string currentUsernameTurn = string.Empty;

private GameSide playerSide;

private bool isMyTurn;

public GameSide PlayerSide { get { return playerSide; } }

public bool IsMyTurn { get { return isMyTurn; } }

public string Username { get { return username; } }

public string RealUsername { get { return realUsername; } }

public string RealEnemyUsername { get { return realEnemyUsername; } }

public Dictionary<string, object> Data { get { return data; } set { data = value; } }

private void Awake() {

DontDestroyOnLoad(this);

Init();

}

private void Init() {

listener = new Listener();

rooms = new List<string>();

data = new Dictionary<string, object> {

{ "Password", "12345" }

};

WarpClient.initialize(Globals.API\_KEY, Globals.SECRET\_KEY);

WarpClient.GetInstance().AddConnectionRequestListener(listener);

WarpClient.GetInstance().AddChatRequestListener(listener);

WarpClient.GetInstance().AddUpdateRequestListener(listener);

WarpClient.GetInstance().AddLobbyRequestListener(listener);

WarpClient.GetInstance().AddNotificationListener(listener);

WarpClient.GetInstance().AddRoomRequestListener(listener);

WarpClient.GetInstance().AddZoneRequestListener(listener);

WarpClient.GetInstance().AddTurnBasedRoomRequestListener(listener);

DontDestroyOnLoad(WarpClient.GetInstance());

username = System.DateTime.UtcNow.Ticks.ToString();

}}

In cele ce au fost redactate mai sus au fost prezentate doar cateva din cele mai importante clase. In ceea ce urmeaza o sa fie prezentata diagrama de clase a acestui joc :

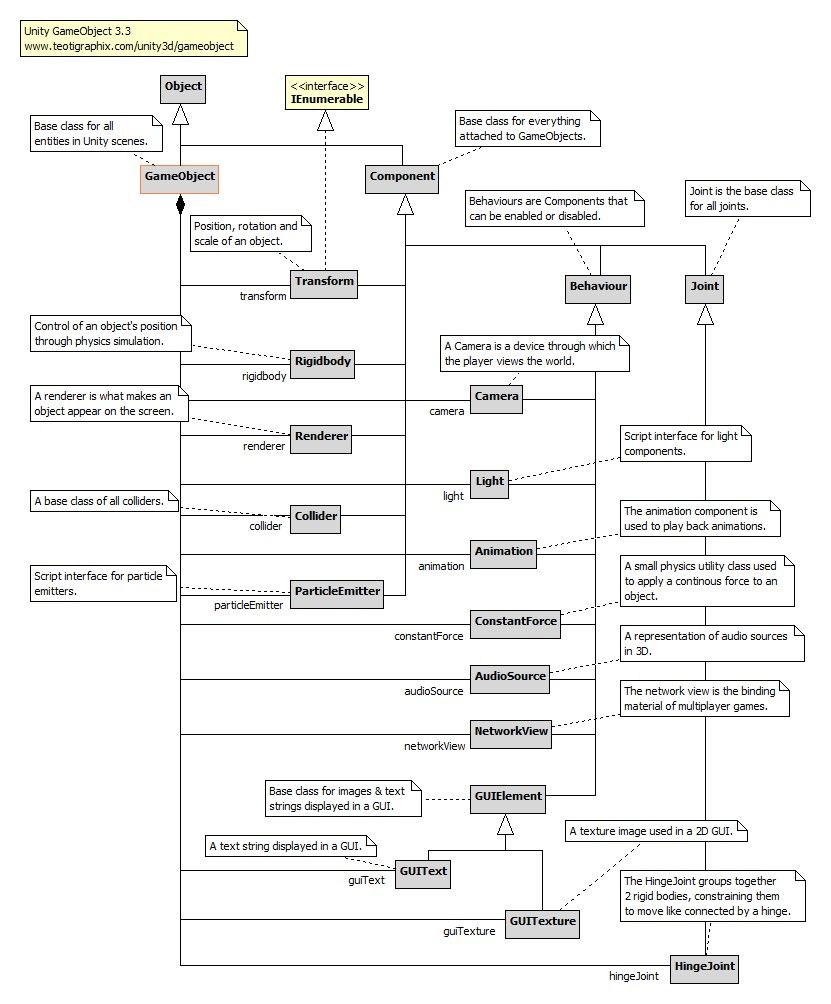
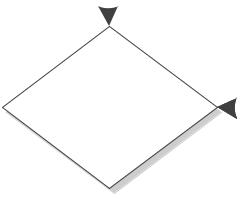


Diagrama care prezinta actiunea AI-ului este prezentata in ceea ce urmeaza :

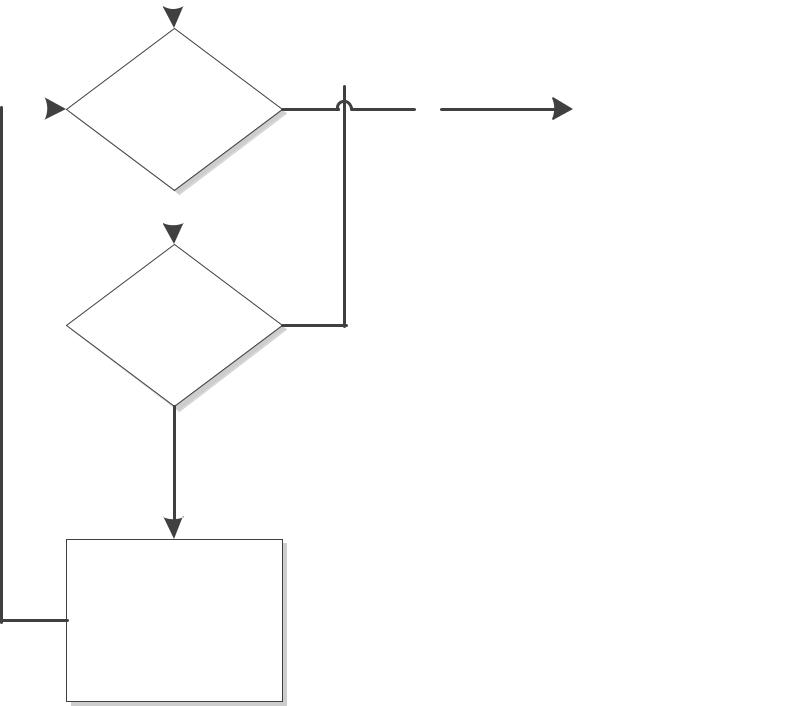
Start

No

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  | Set status to active | |  |  | Did the player |  |
|  | And spawn me | |  |  | activate me? |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | Despawn |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | And set status to |  |  | Yes |  |  |  | Am I dead? | | | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | inactive |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | No | |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | |  |  |
|  |  |  |  |  |  |  |  | Am I in range of | | | |  | Yes | |  | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  | No |  | Move To Player | |
|  |  |  |  |  |  |  |  | the player? | | | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



Am I taking

damage?

Attack Player

**Overview of Functional Requirements**

La nivelul cel mai de bază, jocul va atrage perspectiva utilizatorilor asupra jocului pe monitor, unde jucătorul poate folosi apoi mouse-ul și tastatura pentru a emite comenzi la diferite unități de pe tabla de joc. După primirea unei comenzi, unitatea va efectua acțiunile corespunzătoare. Jocul se va încheia când regele inamicului a fost învins, un jucător se preda, sau ambii jucători sunt de acord cu o remiză.  
  
Mai specific, am identificat următoarele povesti de utilizator:

|  |
| --- |
| • Utilizatorul poate muta o piesă într-o altă locație făcând click pe mouse pentru a apuca piesa și a o trage într-o locație nouă. Fiecare piesă poate avea modele specifice de mișcare pe care trebuie să le respecte sau dacă mișcarea nu va fi luată în considerare.  • Utilizatorul va putea să atace o altă piesă fie mutând în aceeași locație cu piesa inamică, fie mutând lângă piesa inamică. Diferitele piese pot avea modele diferite de atac.  • Unele piese pot avea acțiuni speciale pe care le pot efectua (cum ar fi vrăji). Pentru a utiliza aceste acțiuni, jucătorul selectează piesa făcând clic pe ea. Odată selectată, informațiile despre unități vor apărea într-un meniu din partea de jos a ecranului. Acest meniu va afișa acțiunile disponibile pentru unități, iar jucătorul poate face clic pe acțiune pentru ao activa. Dacă acțiunea necesită o țintă, jucătorul poate selecta ținta făcând clic pe ea în zona de redare. Tastele rapide pot fi disponibile pentru acțiunile unității.  • Jucătorul se poate preda în orice moment.  • Când jucătorul câștigă un meci, el va primi puncte pe care le poate utiliza pentru a particulariza sau actualiza piesele existente sau pentru a cumpăra noi piese. Această particularizare va avea loc între meciuri într-un ecran separat.  • Dacă regele tău este învins, pierzi meciul. Dacă învingeți pe regele inamic, câștigați meciul. |



**Product Perspective**

Zombies Stratego este un sistem independent care nu are nevoie de alte legături externe. Această aplicație se referă la orice utilizator care este atras de strategie și jocul minții. În acest joc, o strategie nu este bună sau rea, fiecare strategie are logica sa, dar va câștiga jucatorul care are o strategie mai puternică. După cum sa menționat mai sus, jocul poate fi jucat în LAN prin crearea unei camere unde se pot conecta 2 utilizatori. Fiecare utilizator își poate alege poziția pentru fiecare pion, făcând strategia să se bată. Făcând click pur și simplu pe poziția dorită, utilizatorul va putea plasa pionul în poziția respectivă. Fiecare dintre ei vor avea 200 de dolari, deoarece fiecare zombie va avea un preț bazat pe puterile și abilitățile pe care le are.  
  
Odată cu începerea jocului, fiecare jucător are 30 de secunde pentru a face o mutare validă și în caz contrar va fi efectuată o mișcare aleatorie. Prin mișcare valabilă se înțelege că pionul se deplasează într-o poziție valida în care nu există nici un pion al oponentului. Odată ce ajungi la peretele adversarului, strategia trebuie mutată pentru a ajunge la steag și a cuceri teritoriul.  
  
Ca sistem de operare Zombies-Stratego va fi rulat pe Windows. Portarea pe o altă platformă nu ar trebui să fie complicată, deoarece nu există legături externe care să întrerupă sau să pună o taxă grea pe acest port.  
  
Aplicația va fi implementată utilizând Unity 3D.