Quixel Texel AI

Un approccio intelligente all'environmental design

Docente

Fabio Palomba

Studenti

Angelo Antonio Prisco e Giovanni Carbone

Università degli Studi di Salerno

Documentazione del progetto sviluppato per il corso di Fondamenti di Intelligenza Artificiale

https://github.com/Giovannicar201/Quixel_Texel

2024

Indice

1 Introduzione

Il mondo dello sviluppo videoludico, in particolar l'ormai vasto mercato indie, risulta ancora oggi un mercato critico, una realtà in cui difficilmente ci si può considerare al sicuro. Sia piccole che grandi aziende sono costrette a rispettare scadenze e pretese sempre meno realistiche. Se da un lato, però, le grandi aziende riescono a far fronte a tali criticità, date le loro ingenti risorse, lo stesso non può essere detto per le aziende minori, le quali, con le poche risorse a disposizione, spesso finiscono per essere prima chiuse, e poi dimenticate. Il rispetto delle scadenze è, per un'azienda indie, una necessità, e riuscire a risparmiare tempo e soldi nell'esecuzione del loro lavoro è l'unica soluzione.

Le attività che richiedono maggiore attenzione nello sviluppo di un videogioco sono due, il game design e l'environmental design. Attività di progettazione che richiedono enormi quantità di tempo, al fine di poter produrre e valutare quante più scelte e prototipi possibili. Tali processi possono quindi rallentare il lavoro di un'azienda indie, inficiando la qualità del prodotto finale.

È quindi necessario individuare una criticità specifica, la quale, se risolta, può agevolare l'operato di un'azienda indie. Una delle tipologie di videogiochi maggiormente approcciata da questo tipo di aziende, data la sua semplicità, è quella dei videogiochi 2D top-down basati su tile-map, ovvero giochi la cui mappa, oltre ad essere gestita come un piano, è gestita anche come una griglia. Generalmente il processo di game design di questi videogiochi è meno complesso rispetto ai videogiochi 3D o a videogiochi 2D ma con caratteristiche diverse. Più complesso è invece il processo di environmental design, siccome spesso si cerca di bilanciare la semplicità di una mappa 2D creando mappe estremamente grandi. Progettare e prototipare questa tipologia di mappe è un processo estremamente lungo, il quale, se fatto manualmente, potrebbe rallentare notevolmente l'intero sviluppo di un videogioco indie.

2 Definizione del problema

2.1 Obiettivi

L'obiettivo che si vuole raggiungere attraverso questo progetto consiste nella realizzazione di un modulo di IA capace di generare porzioni di tile-map sulla base del lavoro svolto da un environmental designer. In particolar modo si vuole far in modo che l'agente si adatti completamente allo stile adottato dall'environmental designer, senza che quest'ultimo si debba preoccupare di dover impostare particolari parametri o impartire specifici comandi, se non quello di selezionare la porzione di mappa da generare. L'IA dovrà più nello specifico riuscire a generare porzioni di mappa che, da un punto di vista della composizione, siano verosimili e soprattutto bilanciate. Generalmente il processo di environmental design parte definendo gli elementi più importanti della mappa, ovvero quelli di maggiore interesse per un giocatore, i quali vengono poi distribuiti su tutta la mappa e collegati tra loro usando elementi di contorno, i quali permettono al giocatore di rimanere concentrato su un punto di interesse. L'IA dovrà quindi essere in grado di replicare tale processo durante la generazione di una porzione di mappa.

2.2 Formulazione P.E.A.S

Di seguito è riportata la formulazione P.E.A.S adattata al nostro modulo di IA.

- **Performance.** La misura di prestazione adottata prevede la massimizzazione della distanza tra gli elementi importanti di una mappa e la minimizzazione della distanza degli elementi poco importanti di una mappa.
- **Environment.** L'ambiente in cui opera l'agente è costituito dallo stato attuale di una mappa, sulla quale potranno essere stati piazzati in precedenza un numero ed una tipologia di tile variabile.
- **Actuators.** L'agente agisce sull'ambiente posizionando un tile in ogni cella della porzione di mappa selezionata dall'environmental designer.
- **Sensors.** L'agente percepisce l'ambiente tramite una virtualizzazione dello stato corrente della mappa, dalla quale estrarrà poi le informazioni necessarie per poter operare.

2.3 Caratteristiche dell'ambiente

Di seguito è riportata una descrizione dell'ambiente.

- **Singolo agente.** Data la natura del problema vi sarà un unico agente a cui sarà affidato il compito di generare una porzione di mappa.
- **Completamente osservabile.** L'agente accedendo allo stato attuale della mappa è in grado di estrarre ogni informazioni riguardo quest'ultima.
- **Deterministico.** Lo stato successivo dell'ambiente è definito unicamente dallo stato attuale e dall'azione eseguita dall'agente.
- **Sequenziale.** L'agente agisce eseguendo una sequenza di azioni.
- **Statico.** L'ambiente non varia mentre l'agente sta operando.
- **Discreto.** L'ambiente ha un numero finito di stati distinti, così come le azioni che l'agente può eseguire.

2.4 Analisi del problema

In questa fase cerchiamo di estrapolare tutti concetti che sono direttamente correlati al problema, siccome tali concetti dovranno essere riportati all'interno del dominio della soluzione.

Sicuramente il concetto di mappa è il primo concetto direttamente correlato al problema in esame, così come lo è anche il concetto di selezione, o di area selezionata, siccome questa coinciderà con la porzione di mappa da generare. Un altro concetto di cui tenere traccia è ciò che è piazzato sulla mappa, ovvero i tile, o entità, e a partire da questi anche l'importanza che essi hanno sulla mappa. Infine è necessario dividere il concetto di entità in altri due concetti, quello di entità univoca e quello di entità piazzata. Questo è necessario perché una stessa entità può essere piazzata in più punti della mappa, per cui sotto un certo punto di vista quelle entità sono uguali, poiché sono la stessa entità, ma allo stesso tempo sono anche diverse, poiché collocate in punti differenti.

Estrapolati tutti questi concetti siamo quindi vicini al poter passare al dominio della soluzione.

3 Tecnologie adottate

Prima di poter giungere alla soluzione effettivamente adottata è bene essere a conoscenza delle tecnologie usate. In particolar modo come linguaggio di programmazione è stato selezionato Java siccome il modulo di IA è integrato all'interno di un sistema basato su Java, inoltre tale sistema gestisce le mappe come file JSON, motivo per cui abbiamo optato per mantenere questo formato come formato di input e di output per il modulo di IA.

4 Soluzione del problema

4.1 Tipologia di IA

Il primo passo da compiere è capire quale tipo di IA può aiutare a risolvere questo problema. Data la natura del problema, sicuramente non si tratterà di un IA basata sui concetti della teoria dei giochi, così come non ci si può rifare ad un ML, sia per la mancanza di dati sia per il tipo di problemi che un ML risolve. Bisogna quindi rifarsi ad un algoritmo di ricerca. Tra gli algoritmi di ricerca possiamo individuare tre macro categorie, quelli non informati, quelli informati e quelli locali.

Gli algoritmi di ricerca non informati non necessitano di informazioni specifiche del problema, per cui possono risultare ottimali in quelle situazioni in cui tali informazioni mancano. Siccome nel nostro caso riusciamo invece ad estrapolare una grande quantità di informazioni relative al problema possiamo scartare l'idea di usare un algoritmo di ricerca non informato. Gli algoritmi di ricerca informati fanno invece uso di informazioni specifiche del problema, ma si focalizzano per lo più sul *come* si è giunti alla soluzione finale, e non su *cosa* effettivamente rappresenti questa soluzione. Gli algoritmi di ricerca locale invece non solo possono far largo uso di informazioni specifiche del problema, ma si focalizzano per lo più sulla soluzione finale, partendo da una soluzione iniziale ed ottimizzandola, per questo motivo un algoritmo di ricerca locale sembra essere la scelta più adatta.

A questo punto è necessario decidere che tipo di algoritmo di ricerca locale utilizzare. Nel caso specifico, trattandosi di un problema relativamente complesso, è necessaria una tipologia di algoritmi in grado di adattarsi bene a tale complessità. Serve quindi una tipologia di algoritmi estremamente flessibili e che garantiscono soluzioni anche sub-ottimali a problemi complessi. Tale descrizione coincide con quella che è la definizione di algoritmo genetico, motivo per cui possiamo definire questo come un punto di partenza.

Rimane quindi da definire quale specifica tipologia di algoritmo genetico utilizzare. Avendo a disposizione molte informazioni relative al problema e dovendo generare porzioni di mappe in un lasso di tempo inferiore a quello impiegato da un environmental designer per crearne una manualmente, abbiamo necessità di un algoritmo genetico in grado di sfruttare tali informazioni e che sia computazionalmente leggero, inoltre non dimentichiamo che il tutto deve avvenire senza che l'environmental designer configuri niente. Definite queste premesse possiamo quindi scartare l'idea di usare un algoritmo genetico interattivo, siccome questo richiederebbe una verifica manuale dell'utente, ed allo stesso tempo tempo non possiamo rifarci ad un algoritmo genetico tradizionale siccome questi spesso richiedono un certo costo computazionale. Gli algoritmi genetici steady state risultano essere quindi la scelta favorita, data la loro possibilità di bilanciare efficienza ed efficacia. Inoltre gli algoritmi steady state si adattano bene a problemi multi-obiettivo, categoria di problemi sotto la quale ricade anche il problema trattato, siccome dobbiamo contemporaneamente assicurarci di massimizzare la distanza tra gli elementi importanti e minimizzare la distanza tra gli elementi poco importanti.

4.2 Algoritmo steady state

Di seguito sono riportati i passi di cui si compone un algoritmo steady state standard.

1. Crea una popolazione iniziale di N individui.
2. Valuta ogni individuo.
3. Seleziona due individui.
4. Esegui il crossover tra gli individui selezionati.
5. Esegui la mutazione sugli individui generati.
6. Valuta gli individui generati.
7. Sostituisci i due peggiori individui della popolazione se questi sono peggiori degli individui generati.
8. Ripeti dal secondo passo se il budget di ricerca non è terminato.

5 Algoritmo risolutivo

Compresi i concetti su cui basare la nostra soluzione e la strategia da seguire, è possibile definire un algoritmo risolutivo. Questo verrà inoltre raffinato con il proseguire delle sezioni siccome di volta in volta verranno esposti i problemi affrontati e le soluzioni proposte. L'algoritmo può essere quindi riassunto in tre semplici fasi.

1. Estrai le informazioni dalla mappa.
2. Individua le entità importanti e poco importanti.
3. Esegui l'algoritmo genetico steady sfruttando le informazioni ottenute.

5.1 Estrazione delle informazioni

Prima di poter eseguire una qualunque operazione dell'algoritmo genetico dobbiamo estrarre quante più informazioni possibili inerenti al problema. Questo è necessario per due motivi, il primo è che tante più sono le informazioni che vengono recuperate tante più sono le informazioni che possono essere usate per risolvere il problema, il secondo è che senza le giuste informazioni non saremo in grado neanche di generare gli individui necessari per compiere il primo passo dell'algoritmo genetico.

5.1.1 Informazioni relative alla mappa

Come già menzionato, il modulo di IA prende in input un file JSON rappresentativo della mappa, ed oltre a tale file richiede due coppie di coordinate, usate per identificare l'area selezionata.

Il primo problema affrontato è che seppur un file JSON risulti estremamente comodo per poter ricevere ed inviare informazioni, quest'ultimo è poco pratico quando si necessita di eseguire continue operazioni matematiche o anche semplici controlli sulle informazioni contenute al suo interno. Siccome una mappa può essere vista come una griglia, e quindi una matrice, il primo passo è stato definire una classe **Parser** che, dato in input un JSON, restituisce una matrice di interi, dove ogni intero positivo rappresenta l'identificativo di un'entità ed ogni intero nullo rappresenta una cella

vuota. A questo punto era possibile estrarre comodamente tutte le informazioni necessarie alle operazioni successive. Tali operazioni di estrazione sono eseguite dalla classe **MappaManager**, la quale oltre a mantenere come variabili d'istanza la mappa e le coordinate della selezione, mantiene anche il numero di entità totali piazzate sulla mappa, la larghezza della selezione, l'altezza della selezione e il numero totale di celle selezionate. Di seguito è riportato un esempio di una mappa valida come input per il modulo di IA.

```
{"mappa":[
{"riga":"0","colonna":"0","id":1,"immagine":"..."},
{"riga":"0","colonna":"1","id":0,"immagine":""},
{"riga":"0","colonna":"2","id":2,"immagine":"..."},
{"riga":"0","colonna":"3","id":3,"immagine":"..."},
{"riga":"1","colonna":"0","id":3,"immagine":"..."},
{"riga":"1","colonna":"1","id":0,"immagine":""},
{"riga":"1","colonna":"2","id":0,"immagine":""},
{"riga":"1","colonna":"3","id":4,"immagine":"..."},
{"riga":"2","colonna":"0","id":5,"immagine":"..."},
{"riga":"2","colonna":"1","id":6,"immagine":"..."},
{"riga":"2","colonna":"2","id":0,"immagine":""},
{"riga":"2","colonna":"3","id":0,"immagine":""},
{"riga":"3","colonna":"0","id":0,"immagine":""},
{"riga":"3","colonna":"1","id":2,"immagine":"..."},
{"riga":"3","colonna":"2","id":2,"immagine":"..."},
{"riga":"3","colonna":"3","id":0,"immagine":""}
]}
```

5.1.2 Informazioni relative alle entità

Estratte le informazioni riguardanti la mappa, abbiamo ancora la necessità di estrarre numerose informazioni relative alle entità piazzate sulla mappa. Abbiamo quindi bisogno innanzitutto di una classe rappresentativa del concetto di entità, per cui è stata definita la classe **EntitaEntity**, la quale mantiene come variabili d'istanza tutte le informazioni relative ad una certa entità. In particolar modo le informazioni da mantenere sono le seguenti.

- L'identificativo dell'entità.
- Il numero di volte che questa compare sulla mappa.
- Il numero di volte che questa compare sulla mappa in percentuale.
- Il numero di volte che questa deve essere piazzata nell'area selezionata.
- Il rimanente numero di volte che questa deve essere piazzata nell'area selezionata.
- Il grado di importanza.

Siccome questi valori devono essere impostati per ogni entità univoca che compare sulla mappa, abbiamo definito una classe **EntitaManager** che esegue questo insieme di operazioni. In particolare l'ultimo ed il penultimo valore richiedono particolare attenzione, siccome il modo in cui questi vengono calcolati va ad impattare sul tipo di risultato che si otterrà e sugli individui che verranno generati successivamente.

Per quanto riguarda il numero di volte che un'entità deve essere piazzata all'interno della selezione si è deciso di basarsi su una proporzione matematica. Per cui se un'entità è piazzata tante

volte sulla mappa allora verrà piazzata, proporzionalmente all'area selezionata, tante volte. Nello specifico il numero di volte che un'entità deve essere piazzata nell'area selezionata sarà pari al risultato del seguente calcolo, definito X il numero di volte che l'entità compare sulla mappa in percentuale e Y il numero totale di celle dell'area selezionata, avremo che il numero di volte che un'entità deve essere piazzata nell'area selezionata sarà pari al risultato del seguente calcolo:

$$\frac{X \cdot Y}{100}\,.$$

Ex. Supponiamo ad esempio che su una mappa composta da 4x4 celle sia piazzata 8 volte l'entità con identificativo 1. Ciò significa che la percentuale con la quale questa entità compare sulla mappa è del 50%. Supponendo che l'area selezionata sia composta da un totale di 6 celle allora il numero di volte che l'entità con identificativo 1 dovrà essere piazzata nell'area selezionata sarà pari al risultato del seguente calcolo:

$$\frac{50 \cdot 6}{100} = 3\,.$$

Per cui il numero di volte che l'entità con identificativo 1 dovrà essere piazzata nell'area selezionata sarà pari a 3. Nel caso in cui sulla mappa dovessero essere piazzate più entità di quante ne possono essere piazzate nell'area selezionata allora, per parte di esse, il numero di volte che dovranno essere piazzate nell'area selezionata sarà pari a 0.

5.2 Definire il grado d'importanza di un'entità

Uno dei fattori che sicuramente può avere maggior impatto sul tipo di risultati ottenuti, è come vengono distinte le entità importanti da quelle poco importanti. Per fare questo partiamo dall'assunzione che se un'entità è piazzata tante volte sulla mappa allora questa, con molta probabilità, non sarà importante, siccome in una qualsiasi composizione vi devono essere pochi elementi importanti, attorno ai quali ruotano tutti gli gli altri elementi.

Prima di proseguire con la soluzione trovata a questo problema è necessario definire i gradi di importanza di un'entità. Nel nostro caso ci è sembrato ideale definire tre gradi di importanza, che è possibile leggere di seguito.

- HIGH LOD.
- MEDIUM LOD.
- LOW LOD.

L'acronimo LOD sta per *Level Of Detail*, per cui un'entità importante sarà definita come HIGH LOD, un'entità mediamente importante sarà definita come MEDIUM LOD ed infine un'entità poco importante sarà definita come LOW LOD.

A questo punto seguiamo un semplice ragionamento. Sicuramente l'entità che percentualmente compare meno volte sulla mappa sarà un'entità importante, ovvero HIGH LOD, mentre l'entità che percentualmente compare più volte sulla mappa sarà un'entità poco importante, ovvero LOW LOD. Così facendo avremo definito un primo range di percentuali. Se a questo punto calcoliamo la media tra la percentuale minima e la percentuale massima saremo in grado di dividere il range in due range minori, per cui saremo in grado di suddividere le entità in base alla percentuale con la quale sono state piazzate sulla mappa in due gruppi.

Siccome serve però poter dividere le entità in tre gruppi di importanza, è necessario eseguire nuovamente questo calcolo, una volta tra la percentuale minore e la percentuale media calcolata al passo precedente e poi tra la percentuale media e la percentuale maggiore, ottenendo così altri due valori medi. Arrivati a questo punto basterà non considerare la prima media calcolata, ma solo quelle calcolate al secondo passo e saremo in grado di dividere il range iniziale in tre range minori, ovvero tre gruppi di importanza differenti.

Non rimane quindi che controllare per ogni entità questa in quale range rientra ed associargli il corretto livello di importanza. Si noti che così facendo il grado di importanza viene assegnato in base allo stato attuale della mappa, quindi è l'environmental designer che in base al numero di volte che piazza un'entità sulla mappa definisce quali sono le entità importanti e quali non lo sono, per cui questo permette al modulo di IA di reputare importanti le entità che l'environmental designer ha reputato importanti, minimizzando i rischi di unire due modi di lavorare sulla mappa incoerenti tra loro.

Ex. Supponiamo ad esempio che l'entità con identificativo 1 sia l'entità piazzata il maggior numero di volte sulla mappa, con una percentuale del 50%. Supponiamo ora che l'entità con identificativo 4 sia l'entità piazzata il minor numero di volte sulla mappa, con una percentuale del 10%. Così facendo saremo in grado di definire un range cha va dal 10% al 50%. Calcolando la media tra questi due valori avremo un punto al centro di questo range, pari al 30%, che divide il range in due gruppi, uno dal 10% al 30%, ed un altro dal 30% al 50%. Se eseguiamo nuovamente questo procedimento prima prendendo tra il 10% e il 30%, e poi tra il 30% ed il 50%, troveremo

altri due punti, pari rispettivamente a 20% e 40%. A questo punto non rimane altro da fare se non ignorare la prima media calcolata, siccome avremo diviso il range iniziale in tre gruppi, il primo dal 10% al 20%, il secondo dal 20% al 40% ed il terzo dal 40% al 50%. A questo punto per ogni entità basterà controllare il gruppo di appartenenza in base alla percentuale con la quale questa è stata piazzata per definire il suo grado di importanza.

5.3 Dettagli dell'algoritmo genetico steady state

Come visto nelle sezioni precedenti, un algoritmo genetico steady state si compone a sua volta di diverse fasi. Per cui è necessario focalizzarsi su ogni fase ed identificare i motivi delle scelte prese.

5.3.1 Codifica dell'individuo

Il primo passo è quello di codificare un individuo, ovvero individuare una struttura dati e delle caratteristiche che rappresentino l'individuo stesso e che permettano di eseguire diverse operazioni, quali valutazione, selezione, crossover e mutazione. In questo caso sia individuare che codificare un individuo è stato relativamente banale, siccome un individuo deve rappresentare la porzione di mappa da generare e può quindi essere codificato come una matrice di interi di grandezza pari all'area selezionata dall'environmental designer. Ogni gene dell'individuo rappresenterà quindi l'identificativo di un'entità.

5.3.2 Vincoli dell'individuo

I vincoli identificati sono due, e sono quelli riportati di seguito.

- Un individuo non deve avere celle in cui non sono piazzate entità.
- Un individuo deve contenere tante volte un'entità quante sono le volte che questa deve essere piazzata nell'area selezionata, valore calcolato attraverso la proporzione presentata nelle sezioni precedenti.

Il motivo del primo vincolo è che, se un environmental designer decide di utilizzare un modulo di IA per poter generare una porzione di mappa si aspetta che questa venga generata completamente e non parzialmente. Il motivo del secondo vincolo è che il modulo di IA deve mantenere uno stile coerente con lo stile adottato dall'environmental designer, rispettando cosa l'environmental designer reputa importante o poco importante.

5.3.3 Obiettivi

Come già menzionato nelle sezioni precedenti, gli obiettivi da raggiungere sono due, e sono quelli riportati di seguito.

- Massimizzare la distanza tra le entità importanti.
- Minimizzare la distanza tra le entità poco importanti.

5.3.4 Generazione della popolazione iniziale

Un algoritmo genetico steady state per risultare efficace presuppone che gli individui iniziali rappresentino già un discreta soluzione. Riuscire a generare individui quantomeno discreti significa far leva sui due obiettivi da raggiungere senza però complicare eccessivamente la generazione di tali individui. La scelta è stata dunque quella generare individui che quantomeno non presentassero troppe entità importanti vicine tra loro. Procedere in maniera totalmente casuale non era quindi possibile, poiché la probabilità che non venissero generati gruppi di entità importanti era un fattore notevolmente influenzato sia dallo stato della mappa che dalla dimensione dell'area selezionata. Per questo motivo si è optato per calcolare un gap costante tra un'entità importante e quella successiva, per calcolare questo gap è stata utilizzata la seguente formula, definito X il numero totale di celle dell'area selezionata e Y il numero di volte che un'entità i importante deve essere piazzata, il gap sarà dato dal risultato del seguente calcolo:

$$\frac{X}{\sum_i Y_i}.$$

5.3.5 Valutazione dell'individuo

Data la presenza di due obiettivi da raggiungere si ha la necessità di definire due funzioni fitness distinte, le quali dovranno poi essere combinate. La scelta di combinare le due funzioni di fitness piuttosto che usare altre tecniche come il fronte di Pareto, è data dal fatto che queste tecniche per quanto utili non risolvono in maniera assoluta il problema, siccome nel caso del fronte di Pareto andrebbe poi definita una funzione di preferenza, rischiando di dare troppa importanza ad una funzione di fitness rispetto all'altra. L'obiettivo è quindi quello di favorire individui che contemporaneamente cercano, anche se non in maniera ottimale, di soddisfare entrambe le funzioni di fitness piuttosto che individui ottimi ma che soddisfano bene solo una delle due funzioni.

Inizialmente abbiamo quindi provato a combinare le due funzione linearmente, assegnando la stessa importanza ad entrambe. Questo ci ha permesso di giungere ad un'unica funzione di fitness espressa dalla seguente formula, definita f la funzione di fitness che mira a massimizzare la distanza tra gli elementi importanti e definita g la funzione di fitness che mira a minimizzare la distanza tra gli elementi poco importanti, il risultato delle funzioni di fitness combinate sarà pari al risultato del seguente calcolo:

$$\frac{f \cdot 50 + g \cdot 50}{100}.$$

Questa scelta però si è rivelata solo parzialmente corretta, siccome diversi individui venivano considerati ottimi anche se di fatto molte entità importanti erano poi posizionate in modo tale da creare gruppi relativamente coesi. Per questo motivo abbiamo quindi deciso di assegnare alle due funzioni di fitness due livelli di importanza simili, in maniera tale da limitare questo problema, giungendo ad avere come risultato della combinazione delle funzioni fitness il risultato del seguente calcolo:

$$\frac{f \cdot 60 + g \cdot 40}{100}.$$

A questo punto non rimane che definire come sono state calcolate le distanze tra le entità importanti e le entità poco importanti. Per ora concentriamoci sulle entità importanti. In questo caso il primo obiettivo è riuscire ad avere un visione d'insieme di come sono disposte le entità importanti e di quanto queste siano distanti tra loro. Per fare questo abbiamo deciso di definire un

grafo non orientato, pesato e completamente connesso, avente come nodi tutte le entità importanti e come pesi degli archi la distanza euclidea tra la posizione dei nodi dell'arco. A questo punto non rimaneva altro che ridurre questo grafo non orientato, pesato e completamente connesso ad un valore numerico utilizzabile per misurare la distribuzione delle entità, per fare questo, siccome dovevamo massimizzare la distanza tra le entità importanti abbiamo ritenuto ideale utilizzare l'algoritmo di Kruskal per poter ricavare il maximum spanning three dal quale poi calcolare la somma dei pesi degli archi che lo componevano.

Capito il modo di misurare la distribuzione delle entità importanti abbiamo deciso di fare lo stesso per le entità poco importanti. Abbiamo quindi deciso di creare un secondo grafo non orientato, pesato e completamente connesso avente come nodi le entità poco importanti e come pesi degli archi la distanza euclidea tra la posizione dei nodi dell'arco. A differenza del primo grafo abbiamo però dovuto utilizzare Kruskal per poter ricavare non più il maximum spanning three ma il minimum spanning three, siccome dovevamo minimizzare la distanza tra le entità poco importanti. Infine a partire dal minimum spanning three ricavato abbiamo poi calcolato la somma dei pesi degli archi che lo componevano.

5.3.6 Selezione

Riguardo il processo di selezione la tecnica definita *Truncation* ci è parsa quella più adatta al nostro caso. In realtà il processo di selezione rispetto al crossover ed alla mutazione è stato il processo meno attenzionato, in quanto con molte probabilità anche altre tecniche di selezione sarebbero state valide e non avrebbero influito particolarmente sul risultato finale.

5.3.7 Crossover

Riguardo il processo di crossover la tecnica definita *Uniform* ci è parsa quella più adatta al nostro caso. Attraverso tale tecnica ogni gene i-esimo dell'individuo generato è scelto casualmente tra i geni i-esimi dei genitori. Il problema riscontrato con questo metodo è che questo, nella sua versione standard, generava nella stragrande maggioranza dei casi individui che non rispettavano il secondo vincolo. Per tale motivo abbiamo definito una variante di questa tecnica in grado di generare individui validi. Per fare questo è stato necessario inserire un controllo inerente al numero di volte che il gene i-esimo del primo genitore veniva piazzato. Quando un gene viene piazzato per un numero di volte pari al numero di volte che l'entità doveva essere piazzata nell'individuo da generare, la probabilità che l'individuo erediti ancora quel gene non è più pari al 50% ma scende a 0.

Anche dopo l'introduzione di questa variante, seppur gran parte degli individui risultavano validi, vi erano comunque particolari individui che una volta aver effettuato il crossover, generavano individui non validi, i quali portavano, con l'avanzare delle generazioni, un numero sempre maggiore di difetti genetici. Per questo motivo abbiamo definito una funzione di recovery di un individuo non valido, la quale permette, operando sui geni dell'individuo, di trasformare un individuo non valido in uno valido, in maniera tale da non introdurre individui non validi all'interno della popolazione.

5.3.8 Mutazione

Riguardo il processo di mutazione sono state sperimentate due tecniche. La prima tra queste consisteva in uno *Swap* di ogni entità importante di un individuo con una delle entità delle celle adiacenti. Tale tecnica si è però rivelata sub-ottimale in quanto vi erano spesso casi in cui più entità importanti venivano traslate nella stessa direzione. In questi casi anche se la posizione di tali entità cambiava la distanza tra di esse rimaneva la medesima, per cui la valutazione dell'individuo non subiva una modifica tale da permettere di esplorare un insieme più vasto di soluzioni.

A questo punto ci è sembrato ideale provare ad applicare come tecnica di mutazione sempre lo *Swap*, ma questa volta tra due celle scelte in maniera casuale dall'individuo. Tale soluzione non solo ha permesso di semplificare il processo di mutazione ma ha anche permesso di generare individui che, proprio grazie a questa mutazione, risultavano generalmente migliori rispetto a quelli di partenza.

5.3.9 Numero di individui iniziali e budget di ricerca

Il numero di individui che abbiamo ritenuto ideale dati gli obiettivi da raggiungere è pari a 30. Tale valore deriva da una serie di valutazioni che miravano a comprendere quanto fossero bilanciati gli aspetti legati all'efficacia e all'efficienza.

Per quanto riguarda invece il budget di ricerca abbiamo optato per una stopping condition basata sul numero di iterazioni, il quale è stato impostato a 10. Tali scelte sono però discusse anche nella penultima sezione del documento, siccome è necessario discutere di tali scelte in maniera più dettagliata.

6 Risultati

A valle dello sviluppo del progetto abbiamo ritagliato del tempo per poter verificare l'efficacia del modulo di IA sviluppato. Di seguito sono riportate una serie di informazioni circa il risultato di queste verifiche.

Prima di tutto è bene sottolineare che tali sperimentazioni sono state eseguite su una macchina avente le seguenti caratteristiche, inoltre sono stati usati come browser sia Chrome che Opera, mentre per quanto riguarda l'ambiente di esecuzione del progetto, quest'ultimo è stato eseguito all'interno di Intellij IDEA 2023.1.2.

- Windows 10 Pro come sistema operativo.
- AMD Ryzen 5 2600X Six-Core Processor 3.60 GHz come processore.
- 16,0 GB di RAM DDR4.
- NVIDIA RTX 2060 6,0 GB GDDR6 come scheda video.

Al fine di valutare l'efficacia del sistema sono state scelte quattro persone che non hanno mai avuto modo, fino al momento della valutazione, di provare il sistema. A queste persone è stato chiesto di creare una prima mappa composta da 16x16 celle e di riempirla cercando di creare uno scenario quanto più verosimile possibile, il tutto mentre venivano cronometrate. Inoltre è importante dire che ad ognuna di queste persone è stato assegnato lo stesso insieme di entità da poter usare per poter popolare la mappa. Da queste prime misurazioni i dati raccolti sono i seguenti.

	A	B	C	D
Tempo impiegato dall'utente	4.95 min	3.01 min	3.41 min	4.12 min

Eseguita questa prima fase di raccolta dati, è stato chiesto alle persone di creare una nuova mappa, sempre composta da 16x16 celle, ma di riempire solo le prime due righe e di lasciar generare la restante parte della mappa al modulo di IA, ovvero una porzione di mappa composta da 16x14 celle. In questo caso è stato misurato il tempo totale per completare la mappa, un tempo parziale durante il quale la persona ha popolato le prime due righe della mappa e un tempo parziale durante il quale l'IA ha generato la porzione di mappa indicata. Di seguito sono riportati i dati raccolti.

	A	B	C	D
Tempo totale	1.22 min	42 s	2.02 min	1.33 min
Tempo impiegato dall'utente	48 s	20 s	48 s	23 s
Tempo impiegato dall'IA	49 s	22 s	1.04 min	57 s

A questo punto è stato calcolato per ogni persona il tempo, in percentuale, risparmiato per completare la mappa. Di seguito sono riportati i risultati di tali operazioni.

	A	B	C	D
Tempo risparmiato in percentuale	75,36%	86,05%	41,77%	67,72%

Infine è stato valutato l'operato del modulo di IA in contesti in cui è stato chiesto di generare porzioni di mappe più ampie di una porzione composta da un totale di 256 celle. In queste situazioni il modulo di IA sembra non riuscire a trovare una soluzione in tempi ragionevoli. Con molte probabilità questo è dovuto a diversi fattori.

Il primo tra tutti la complessità nel dover creare ed operare su due grafi completamente connessi che, soprattutto in questi casi, raggiungono dimensioni considerevoli. Per questo motivo si potrebbe pensare di procedere in due modi, o modificando il numero di individui della popolazione e modificando il numero di iterazione rappresentativo del budget di ricerca, o definire un budget di ricerca ibrido, dove oltre al numero di iterazioni viene preso in considerazione anche un tempo massimo, dopo il quale l'algoritmo deve restituire la migliore soluzione trovata.

7 Considerazioni finali

Possiamo quindi concludere, date le diverse considerazioni fatte lungo tutto il documento, che gli obiettivi definiti durante la fase iniziale sono stati soddisfatti. Ciò però non significa che il progetto si trovi in uno stato tale da non dover subire modifiche, anzi sono molti gli aspetti che possono essere valutati con maggiore attenzione al fine di trovare la giusta combinazione di valori che garantirà il risultato migliore possibile, in particolar modo questo è riferito a come attualmente è gestito i budget di ricerca. Ad ogni modo tale esperienza ha permesso di approfondire quelle che sono le possibilità ed i limiti di un algoritmo genetico, ed inoltre crediamo che una versione raffinata di tale progetto possa realmente fare la differenza per tutti quei team che approcciano la creazione di videogiochi 2D top-down basati su tile-map.