POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Informatica
Scuola di Ingegneria Industriale e dell'Informazione

POLITECNICO
MILANO 1863

Generazione Procedurale di Contenuti Applicata ai Livelli di un Gioco

Relatore: Prof. Pier Luca Lanzi

Tesi di Laurea di:
Roberto Capiotto, matricola 783825

Anno Accademico 2014-2015

A tutte le persone che mi hanno accompagnato fino a qui

Sommario

I videogiochi necessitano di molto tempo per progettare l'ambiente di gioco e per realizzare tutti i contenuti ed è compito del designer progettare degli scenari e delle strategie che rendano il videogioco divertente, coinvolgente e vario.
L'automatizzazione del processo di creazione di contenuti e di livelli di gioco garantirebbe un risparmio di tempi e di risorse economiche.
L'obiettivo di questo lavoro è l'analisi di come è possibile rendere autonomo il processo di creazione dei livelli di un videogioco come *Journey at the Edge of the Universe*, un puzzle game basato sulla fisica.
In particolare ci siamo concentrati sulla implementazione degli algoritmi che generano i livelli del videogioco, analizzandone la struttura e le migliorie che potrebbero essere applicate nel futuro.

Ringraziamenti

Vorrei innanzitutto ringraziare il prof. Pier Luca Lanzi per la sua disponibilità e per l'aiuto che mi ha dato nello sviluppo del videogioco, nella stesura della tesi e in tutto il percorso di studio.
Ringrazio l'Ing. Daniele Loiacono e l'Ing. Mauro Massari per il tempo che mi hanno dedicato e per i preziosi consigli ricevuti per rendere il gameplay del videogioco il più realistico e il più avvincente possibile.

Ringrazio i miei genitori, Claudio e Graziella, che mi hanno dato l'opportunità di vivere al meglio questi anni di formazione. Il loro sostegno non mi è mai mancato e finalmente oggi, dopo tanti sacrifici, raggiungiamo insieme un grande traguardo.
Ringrazio Federica, il mio grande amore, perchè è sempre stata al mio fianco e ha condiviso con me momenti di gioia e di difficoltà. La ringrazio per avermi sempre supportato e per avermi dato ogni giorno le motivazioni per arrivare fino in fondo.
Un grande ringraziamento a Thomas e Gaia che, con i loro sorrisi e il loro affetto, mi sono sempre stati vicini.
Ringrazio i nonni, gli zii e tutti i parenti che mi hanno accompagnato e sostenuto in questo lungo cammino, con un pensiero speciale rivolto a chi oggi non è più qui.

Grazie ad Andrea, Giulia, Luca e Mattia che hanno passato insieme a me una parte del loro corso di Laurea al Politecnico di Milano. Il loro sostegno, anche al di fuori dell'università, mi è stato di grande aiuto.
Ringrazio tutti gli amici che mi hanno regalato momenti di spensieratezza e di felicità e che mi hanno permesso di crescere come persona. Siete davvero tanti e avete tutti un posto riservato nel mio cuore.

Un ringraziamento speciale a Rosetta, Philae e New Horizons. Le loro imprese mi sono state di grande ispirazione nello sviluppo della tematica del gioco.

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

I videogiochi necessitano di molto tempo per progettare l'ambiente di gioco e per realizzare tutti i contenuti ed è compito del designer progettare degli scenari e delle strategie che rendano il videogioco divertente, coinvolgente e vario. Questo lavoro necessita una grande conoscenza del tema e delle meccaniche di gioco e richiede anche l'impiego di tempi lunghi e risorse economiche elevate. L'automatizzazione del processo di creazione di contenuti e di livelli di gioco garantirebbe un risparmio su entrambi questi aspetti.
In questo lavoro abbiamo studiato e sviluppato delle tecniche per automatizzare il processo di creazione dei livelli di un videogioco come *Journey at the Edge of the Universe*, un puzzle game basato sulla fisica. Questo videogioco è stato sviluppato appositamente per questa tesi ed è stato testato sia in una versione per PC che su smartphone Android. In particolare il lavoro si concentra sulla implementazione degli algoritmi che generano i livelli di gioco.

Questo lavoro è strutturato secondo il seguente schema:

Il **Capitolo 2** contiene una descrizione dello stato dell'arte, introduce la Generazione Procedurale di Contenuti (PCG) e la sua evoluzione nel mondo dei videogiochi. In seguito viene presentata una classificazione degli algoritmi di generazione. Successivamente viene analizzata la generazione procedurale di contenuti con algoritmi di ricerca. Infine viene approfondita la PCG applicata ai puzzle game.

Il **Capitolo 3** introduce *Journey at the Edge of the Universe*, il videogioco che ho sviluppato e su cui è incentrato questo lavoro. In particolare analizza gli elementi di gioco, le funzionalità ad essi associate e le varie tipologie di livello che si possono affrontare. In seguito viene presentato il sistema di interfaccia utilizzato dall'utente per interagire con il videogioco. Infine vengono elencati tutti gli strumenti utilizzati durante lo sviluppo del videogioco.

Il **Capitolo 4** analizza gli algoritmi che generano i livelli di gioco e illustra come questi algoritmi sono stati implementati. Inoltre viene analizzato come la generazione procedurale teorica è stata applicata in *Journey at the Edge of the Universe*.

Il **Capitolo 5** conclude il lavoro, riportando le considerazioni finali e i possibili sviluppi futuri.

2 Stato dell'arte

In questo capitolo introduciamo la generazione procedurale di contenuti o *Procedural Content Generation* (PCG) e la sua evoluzione nel mondo dei videogiochi. In seguito viene presentata una classificazione degli algoritmi di generazione. Successivamente viene analizzata la generazione procedurale di contenuti con algoritmi di ricerca. Infine viene approfondita la PCG applicata ai puzzle game.

2.1 Generazione Procedurale di Contenuti

La generazione procedurale di contenuti lascia che sia il computer a creare in modo autonomo alcuni contenuti di un videogioco, come ad esempio livelli, mappe, armi, circuiti, obiettivi, ecc. In origine la PCG nasce per far fronte a limitazioni di memoria; in passato i calcolatori non ne avevano molta a disposizione e la loro potenza computazionale era limitata. Facendo uso della PCG i contenuti venivano generati solo quando se ne aveva il bisogno, senza sprecare inutilmente risorse preziose.
Il primo videogioco a fare uso della generazione procedurale è Beneath Apple Manor (1978), in cui l'obiettivo è recuperare una mela d'oro nascosta in una delle stanze nel seminterrato di un maniero. Ogni livello, composto da stanze, mostri, tesori, passaggi segreti, etc viene generato in maniera casuale all'inizio del gioco. Questo videogioco ha dato inizio a un genere che ha avuto il suo massimo esponente in Rogue (1980) da cui questo genere ha poi preso il nome (*Roguelike*).
Un altro gioco basato sulla PCG è stato Elite (1984), un videogioco a tema spaziale, basato sul commercio interplanetario. L'universo di Elite contiene 8 galassie di 256 pianeti ognuna e la generazione procedurale ha potuto ovviare alle limitate capacità dei processori a 8 bit. L'algoritmo generava una sequenza di numeri che determinavano per ogni pianeta la sua posizione, il suo nome, i prezzi dei suoi beni e altri dettagli. Anche ogni galassia è generata proceduralmente, partendo dalla prima.

Figura 2.1: Beneath Apple Manor

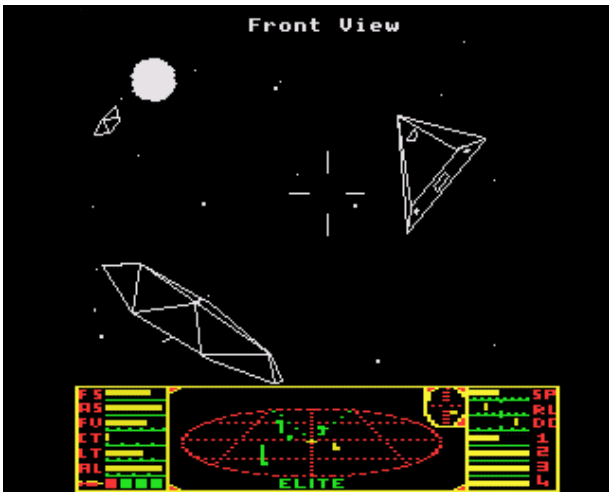


Figura 2.2: Elite

Al giorno d'oggi non si hanno più problematiche di questo tipo, ma nonostante questo la PCG continua ad essere utilizzata per diversi motivi tra cui, come in passato, l'efficienza. Creando i contenuti quando vengono richiesti e non tutti in una sola volta, si ottiene un risparmio sul tempo di computazione e si ha la possibilità di crearne molti di più. Dando al giocatore molto materiale gli si consente di prolungare il gameplay. La generazione procedurale ha anche un impiego nell'industria: la domanda dei videogiocatori è sempre in crescita e le aziende si vedono costrette a pagare sempre di più artisti e programmatori per far fronte a questa richiesta. L'intelligenza artificiale permette ai designer di concentrarsi maggiormente sull'aspetto creativo ed artistico piuttosto che su quello tattico e strategico [6]. All'aspetto strategico ci pensano gli algoritmi che, producendo dei contenuti di buona qualità, permettono di limitare le spese. Infine la generazione procedurale viene utilizzata come supporto alle meccaniche di gioco e vengono quindi creati videogiochi basati interamente su di essa [10]. Se i contenuti vengono generati con buona varietà e in real time diventa possibile creare dei giochi il cui gameplay può essere infinito e che possono essere anche rigiocati infinite volte. Se i contenuti rispecchiano determinati criteri, il giocatore sarà in grado di esplorare la scena e di incontrare ogni volta dei nuovi elementi che renderanno il gioco più attraente.
Esempi di giochi più recenti che fanno uso della PCG sono Diablo (1996) e Diablo 2 (2000) che basano le proprie meccaniche di gioco proprio su di essa, Spore (2008) che genera proceduralmente la musica, i personaggi del gioco e le loro animazioni grafiche [1] e Minecraft (2011), un gioco tridimensionale in cui la PCG genera una serie di cubi che vanno a formare i personaggi e l'ambiente di gioco.

Figura 2.3: Spore

Figura 2.4: Minecraft

Gli algoritmi di generazione procedurale vengono utilizzati all'interno dei videogiochi anche per verificare che essi siano giocabili e divertenti [2]. Uno degli aspetti più importanti da considerare quando si generano i contenuti di un gioco è la qualità dei contenuti stessi. Un indice di qualità riguardante i contenuti è la giocabilità; infatti situazioni di gioco in cui non esistono percorsi che collegano due punti oppure combattimenti contro nemici che non possono essere sconfitti genera un forte senso di frustrazione nel giocatore e questo rende il gameplay impossibile da portare a termine. Sono indesiderati, ma pur sempre accettabili, dei livelli noiosi o troppo facili perchè permettono comunque al giocatore di portare a termine la partita.
Alcuni generatori di livelli hanno implementato dei controlli di giocabilità, come ad esempio il controllo dell'esistenza di percorsi, la misurazione di distanze e l'esistenza di soluzioni. In molti casi risulta utile giocare il livello appena generato per verificare che sia davvero giocabile. Per fare questo è richiesta la costruzione di un agente in grado di giocare.
Gli approcci più diretti che permettono di creare agenti giocatori fanno uso del reinforcement learning e della ricerca ad albero. Il **reinforcement learning** [9] è una tecnica di apprendimento automatico in cui un agente analizza gli input e la situazione dell'ambiente in cui opera per compiere un'azione che genera un determinato output. E' una tecnica abbastanza lenta nell'imparare a giocare bene, ma una volta arrivata a convergenza è un'ottima soluzione. La **ricerca ad albero** può avvenire in maniera ordinata oppure in profondità ed è ottima per giochi semplici e con poche regole, mentre diventa ingestibile per giochi a livelli con uno spazio degli stati piuttosto ampio. La ricerca esaustiva in profondità su tutti gli stati richiederebbe uno sforzo computazionale eccessivo e tempi troppo lunghi.

2.2 Tipologia di algoritmi di Generazione Procedurale di Contenuti

La PCG si è sviluppata nel tempo sotto varie forme, ma fino a poco tempo fa nessuno si era mai occupato di studiarla dal punto di vista scientifico. In [4] gli autori propongono delle classificazioni e delle definizioni a cui attenersi nelle trattazioni sulla generazione procedurale di contenuti.

La prima distinzione riguarda il metodo con cui i contenuti vengono generati: questo può avvenire **online**, quindi durante l'esecuzione, oppure **offline**, cioè durante lo sviluppo del gioco.
La generazione online crea gli elementi ogni qualvolta ne abbiamo il bisogno, distribuendo così la richiesta computazionale nel tempo e evitando di utilizzare la memoria inutilmente. Questa tecnica richiede che l'algoritmo di generazione sia molto veloce, che impieghi una quantità di tempo prevedibile e che il risultato sia soddisfacente. Nel caso della generazione offline invece i contenuti devono essere generati prima dell'avvio del gioco e una volta creati non si ha più la possibilità di modificarli. Esistono anche casi intermedi dove la tecnica online viene unita a quella offline, ad esempio sfruttando informazioni raccolte online dalle ultime partite giocate per poi creare in modalità offline

dei contenuti più adatti a un determinato tipo di giocatore.

La successiva distinzione è relativa al tipo di contenuti generati: essi possono essere **necessari** oppure **opzionali**.
I contenuti necessari sono essenziali per permettere al giocatore di progredire nel gioco e quindi devono essere corretti: se uno di questi contenuti funziona in maniera errata, il giocatore non sarà in grado di andare avanti a giocare. I contenuti opzionali invece possono anche non funzionare a dovere, anzi, lo strano funzionamento può essere parte integrante del gameplay. Il giocatore deve imparare quali elementi può utilizzare e quali no, a patto che gli elementi opzionali siano davvero un'alternativa agli elementi necessari del gioco.

L'algoritmo che genera i contenuti può essere catalogato in base alla sua parametrizzazione. Se l'algoritmo utilizza un vettore di numeri generati casualmente come input, allora l'algoritmo sarà detto di tipo "**random seeds**". Se invece riceve in input un vettore multidimensionale di parametri che specifica alcune regole a cui i dati generati si devono attenere, l'algoritmo sarà di tipo "**parameter vector**". Gli algoritmi random seeds offrono maggiore libertà e i dati generati possono essere molto più vari; al contrario gli algoritmi parameter vector permettono un controllo maggiore sui contenuti generati e le limitazioni possono essere dovute a requisiti legati al design del gioco.

Gli algoritmi possono poi essere distinti tra **generazione stocastica** e **generazione deterministica**. Si può scegliere di utilizzare algoritmi che impiegano più o meno casualità; gli algoritmi deterministici, dati gli stessi parametri in ingresso producono sempre lo stesso contenuto e quindi ci danno una maggiore certezza dei risultati. Gli algoritmi stocastici invece offrono maggiore variabilità; questo può essere un vantaggio in termini di gameplay ma rischiamo di ottenere contenuti di qualità inferiore.

Infine gli algoritmi possono essere differenziati in "**constructive**" e "**generate-and-test**". Gli algoritmi di tipo constructive generano il contenuto una sola volta e l'unica cosa di cui si devono preoccupare è di controllare durante la generazione che quello che è stato generato è di una qualità accettabile. Gli algoritmi di tipo generate-and-test invece hanno un doppio meccanismo di generazione e test: si genera un elemento candidato e lo si sottopone al test per verificare che rispetti determinate caratteristiche. Se il test viene superato, l'elemento candidato diventa effettivo, mentre se il test non viene superato, l'elemento candidato viene distrutto e si ripete da capo l'operazione finchè i contenuti generati non rispettano tutte le regole che vengono imposte.

2.3 Generazione procedurale di contenuti con algoritmi di ricerca

Gli algoritmi di ricerca applicati alla PCG sono uno speciale caso di algoritmo "generate-and-test" introdotto precedentemente. Questo algoritmo ha tre importanti caratteristiche: la prima riguarda il tipo di **rappresentazione** scelta per i contenuti . La seconda caratteristica riguarda la funzione di test, che non si limita ad accettare o a rifiutare un candidato, ma lo valuta assegnandogli uno o più valori reali. Questa funzione viene chiamata **funzione di fitness** e la valutazione da essa assegnata è definita come valore di fitness. La terza caratteristica invece riguarda l'**algoritmo** per la generazione dei nuovi contenuti candidati. La generazione dei nuovi candidati è influenzata dai valori di fitness assegnati ai contenuti valutati in precedenza. Così facendo è possibile creare nuovi contenuti con valore di fitness più elevato rispetto ai precedenti.

2.3.1 Rappresentazione

I contenuti generati possono essere espressi mediante due rappresentazioni: la prima è utilizzata dall'algoritmo di ricerca e viene definita come **genotipo**. Il genotipo è un vettore di parametri che codifica il contenuto. La seconda rappresentazione viene utilizzata dalla funzione di fitness ed

è definita come **fenotipo**. Il fenotipo è la rappresentazione finale del contenuto. Ad esempio il genotipo può essere composto dalle istruzioni che creano un livello, mentre il fenotipo è il livello stesso. E' importante distinguere le rappresentazioni in due tipi di codifica: diretta e indiretta. Nella codifica **diretta** ogni elemento del genotipo è linearmente dipendente dal fenotipo. Nella codifica **indiretta** invece genotipo e fenotipo hanno una relazione non lineare e creare il fenotipo richiede una complessità computazionale maggiore rispetto al caso della codifica diretta.
Una rappresentazione molto usata per il genotipo prevede l'uso di un vettore di numeri reali. Essi infatti sono analizzati semplicemente da diversi algoritmi e sono preferibili rispetto a codifiche non del tutto usuali. Questi vettori devono tenere conto della dimensionalità e della località delle informazioni. La **dimensionalità** del vettore utilizzato per rappresentare il contenuto può essere un problema perchè se è troppo ridotta le informazioni potrebbero non rappresentare in maniera completa il contenuto, mentre se è troppo elevata l'algoritmo potrebbe richiedere troppo tempo prima di arrivare a convergenza. E' importante avere anche una alta **località** delle informazioni perchè una piccola variazione del genotipo genera una piccola variazione nel fenotipo e di conseguenza una piccola variazione nel valore di fitness. Secondo un esempio concreto [5], un livello per un platform 2D come *Super Mario Bros* può avere una rappresentazione diretta come una griglia in cui i contenuti di ogni cella sono specificati separatamente oppure indiretta come la raccolta di diverse configurazioni di muri e spazi vuoti e la lista di come sono distribuiti nel livello. La rappresentazione potrebbe essere ancora più indiretta ad esempio come un elenco delle proprietà desiderate oppure se formata da numeri casuali.

2.3.2 Funzione di fitness

Quando il contenuto candidato viene generato è necessario valutarlo usando una funzione di fitness. Questa funzione valuta il grado di affidabilità degli elementi analizzati rispetto a quelli desiderati assegnando un valore reale ad ogni elemento candidato. Chi realizza la funzione deve decidere che cosa deve essere ottimizzato e il modo in cui formalizzarlo.
Le funzioni di fitness possono essere classificate in base all'approccio con cui i candidati vengono analizzati:

Funzione di fitness **diretta**: si estraggono alcune caratteristiche interessanti dal contenuto generato e in base a queste si calcola un valore di fitness per valutarne la qualità. Degli esempi possono essere il numero di percorsi per uscire da un labirinto oppure la dispersione spaziale delle risorse su una mappa in un gioco di strategia.
Una funzione di fitness può essere a sua volta basata sulla teoria oppure sui dati. La funzione di fitness **basata sulla teoria** viene creata in base alle esperienze e a cosa vogliono i giocatori di un determinato tipo di videogioco. La funzione di fitness **basata sui dati** viene creata sulla base delle informazioni raccolte tramite questionari o strumenti simili; i dati che si ottengono permettono di creare un modello che aiuta a formalizzare stati emozionali del giocatore come divertimento, frustrazione, ansia e competizione.

Funzione di fitness **basata sulla simulazione**: valutiamo il contenuto sulla base dei dati raccolti da uno o più agenti artificiali che, mossi dall'intelligenza artificiale, esplorano l'ambiente di gioco e testano il contenuto generato. Al termine del test vengono salvati dei dati, come ad esempio il tempo impiegato per vincere o il punteggio ottenuto, che vengono poi utilizzati per il calcolo della funzione di fitness.
Possiamo inoltre suddividere le funzioni di fitness basate sulla simulazione in statiche e dinamiche. La funzione di fitness si dice **statica** se l'agente non modifica il suo comportamento durante l'interazione con l'ambiente di gioco. La funzione di fitness è invece **dinamica** se l'agente, durante il test sul contenuto, modifica il suo comportamento e di questo cambiamento ne viene tenuto conto nel calcolo del valore di fitness. Ad esempio se l'implementazione dell'agente fa uso di algoritmi di

apprendimento, il valore di fitness dipenderà da quanto tempo impiegherà l'agente a imparare a giocare.
Va tenuto conto che una funzione basata sulla simulazione richiede uno sforzo computazionale molto maggiore rispetto a una funzione diretta. In quest'ultima vengono svolti pochi calcoli per ogni candidato, mentre in quella basata sulla simulazione si deve attivare un agente che testa ogni candidato, impiegando molte risorse.

Funzione di fitness **interattiva**: l'utente finale viene coinvolto nella raccolta dei dati utili per il calcolo dei valori di fitness. L'interazione tra i giocatori reali e i contenuti generati permette di raccogliere dei dati durante il gameplay. I dati possono essere raccolti in modo implicito oppure esplicito. La raccolta di dati **implicita** può avvenire analizzando i comportamenti del giocatore: quante volte interagisce con un elemento, quando decide di chiudere il gioco e come interagisce fisicamente con il controller del videogioco muovendolo, agitandolo, etc. La raccolta di dati **esplicita** può avvenire tramite sondaggi o questionari e può risultare fastidiosa perchè interrompe il gameplay. Quella implicita è meno invasiva di quella esplicita ma al tempo stesso è meno precisa perchè i dati raccolti possono essere incompleti o poco affidabili.

2.3.3 Algoritmi di ricerca

Dopo aver fissato una rappresentazione e una funzione di fitness si deve scegliere un algoritmo di ricerca. Un algoritmo di ricerca serve a migliorare la qualità dei contenuti generati utilizzando i valori di fitness a disposizione. Spesso si sceglie di utilizzare degli algoritmi evolutivi che implementano meccaniche basate sull'evoluzione biologica come la **selezione**, la **riproduzione**, la **combinazione** e la **mutazione**. Inizialmente l'insieme di candidati, detto anche popolazione, viene generato in maniera casuale. Dopo aver applicato la funzione di fitness ad ogni elemento candidato, si passa alla fase di selezione in cui i contenuti con i valori di fitness più alti vengono ricopiati nella nuova popolazione, mentre i contenuti con i valori di fitness più bassi vengono scartati. I contenuti selezionati vengono poi combinati tra loro con una certa probabilità e possono subire anche un processo di mutazione. Questo procedimento serve a migliorare la popolazione degli elementi candidati e viene ripetuto finchè la popolazione non raggiunge un certo valore medio di fitness oppure dopo aver raggiunto un certo numero di generazioni.

2.4 Generazione procedurale di contenuti applicata ai puzzle game

La generazione procedurale applicata ai puzzle game [8] non richiede soltanto che il gioco sia risolvibile, ma che lo sia sotto determinate condizioni che riguardano il grado di apprendimento del giocatore. I generatori di contenuti sono spesso sia di tipo constructive che di tipo generate-and-test. Gli algoritmi constructive garantiscono, sulla base di come è costruito l'algoritmo, che alcune proprietà verranno rispettate mentre altre proprietà possono soltanto essere forzate modificando i valori da fornire dall'algoritmo. Gli algoritmi generate-and-test, invece, tendono ad automatizzare il processo di generazione e spesso sono implementati mediante l'uso di algoritmi genetici. Anche questi algoritmi però richiedono un intervento umano che decide quando i contenuti generati sono sufficientemente adatti per essere usati nel gioco.
La generazione dei livelli nei puzzle game [6] viene usata per creare un grande numero di livelli che un essere umano non sarebbe mai in grado di creare manualmente in un tempo ragionevole. Le linee guida che deve seguire la generazione ci assicura che tutti i livelli siano giocabili e soddisfacenti per ogni tipo di giocatore.

Sono state svolte molte ricerche riguardo alla generazione di livelli per puzzle game e la prima in assoluto che ne tratta riguarda la generazione di livelli risolvibili per il gioco *Sokoban* [11]. L'algoritmo utilizzato per verificare la giocabilità è il *Breadth First Search* (BFS) e risulta che per ogni 500 livelli generati, solo 14 sono considerati dei buoni livelli perchè la sequenza di soluzione risulta

sufficientemente breve.

In [12] viene generato un editor di livelli che può essere usato per generare nuovi livelli oppure per testare la giocabilità di livelli già esistenti del gioco *Fruit Dating*. I nuovi livelli vengono generati prima creando il layout del livello e poi disponendo gli elementi di gioco secondo delle regole prestabilite. A questo punto viene cercata una soluzione usando un algoritmo simile al BFS. Questa tecnica impiega meno di due minuti per generare un livello, ma non ha nessun controllo sulla difficoltà del livello generato.

L'algoritmo che opera sui livelli del gioco *Refraction* [8] prima genera una possibile soluzione, poi la converte in formato geometrico e infine ne verifica la giocabilità. Il tutto è stato realizzato da due diversi algoritmi, Algorithmic Approach e Answer Set Programming (ASP). Risulta che ASP è più rapido, mentre Algorithmic Approach genera livelli più differenti tra loro.

Gli ultimi due studi che andiamo ad analizzare sono stati applicati a due videogiochi piuttosto famosi. Il primo è stato fatto su un clone di *Cut The Rope*, un gioco pubblicato nel 2010 da ZeptoLab per dispositivi iOS e Android. Il secondo invece su un clone del famosissimo *Super Mario Bros*, pubblicato dalla Nintendo nel 1985.

2.4.1 Cut The Rope: Play Forever

Cut The Rope: Play Forever è un clone del gioco *Cut The Rope* ed è stato realizzato con dei livelli generati proceduralmente. I livelli vengono generati usando una grammatica evolutiva [2 , 3], realizzata in modo tale da garantire che ogni oggetto sarà presente almeno una volta. La funzione di fitness ci fornisce informazioni riguardo alla giocabilità del livello ed è calcolata secondo un approccio basato sulla simulazione. Il livello viene giocato da un agente che valuta, di volta in volta, le azioni eseguibili e, realizzando un albero, verifica se è possibile completare il livello in un numero limitato di azioni.

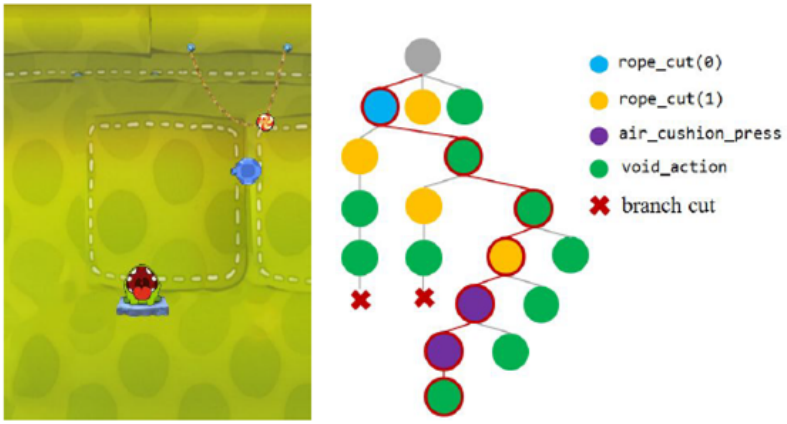
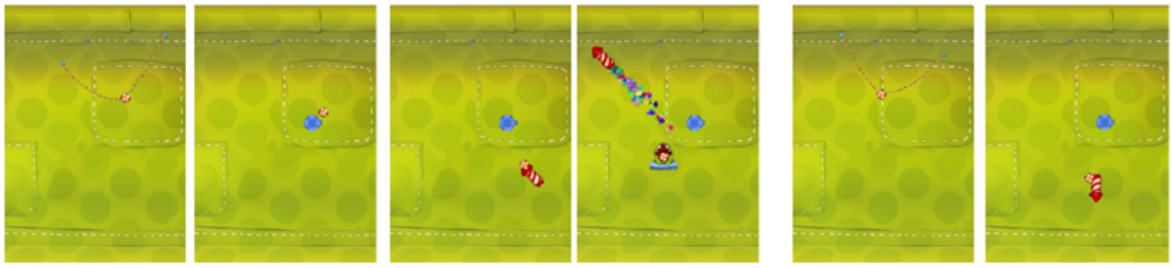


Figura 2.5: Cut The Rope: Albero delle azioni

Recentemente è stata introdotta una nuova tecnica chiamata Progressive Approach [13] che può essere applicata a qualunque tipo di gioco per ridurre i tempi di generazione. Questa tecnica utilizza una grammatica evolutiva per generare la timeline degli eventi e in seguito viene impiegato un agente intelligente per valutare la timeline del livello. Il Progressive Approach è stato applicato su *Cut The Rope: Play Forever* e, confrontando i nuovi risultati con quelli ottenuti in [2 , 3], si nota una forte diminuzione dei tempi di generazione e una migliore varietà e usabilità dei contenuti generati.



(a) Progressively designing a successful mapping. (b) Unsuccessful mapping.

Figura 2.6: Cut The Rope: (a) Livello costruito correttamente e (b) Livello costruito non correttamente

2.4.2 Infinite Mario Bros

Infinite Mario Bros è un clone del più famoso videogioco *Super Mario Bros.* Le regole dei due giochi sono identiche ma questa versione, sviluppata nel 2008 da Markus Persson, è di pubblico dominio e quindi liberamente utilizzabile da chiunque. Durante la *Mario AI Championship* del 2010 si è tenuta la competizione "Level Generation Track" [7] che consisteva nel realizzare un software in grado di creare dei livelli di *Infinite Mario Bros* adatti ad essere giocati da esseri umani.
Ogni partecipante ha realizzato un proprio generatore che esegue algoritmi strutturati in modo diverso tra loro; andiamo ad analizzare i primi due classificati.
Al primo posto si è classificato Ben Weber con il *Probabilistic Multi-Pass Generator*. Questo generatore crea un livello di base e lo modifica fino a 6 volte aggiungendo ad ogni iterazione dei nuovi elementi come terreni, colline, tubi, nemici, blocchi e monete sulla base di un calcolo probabilistico. Questo algoritmo è stato realizzato a livelli di questo genere, ma l'idea di creare un livello di base e poi arricchirlo proceduralmente può essere estesa anche ad altri generi.

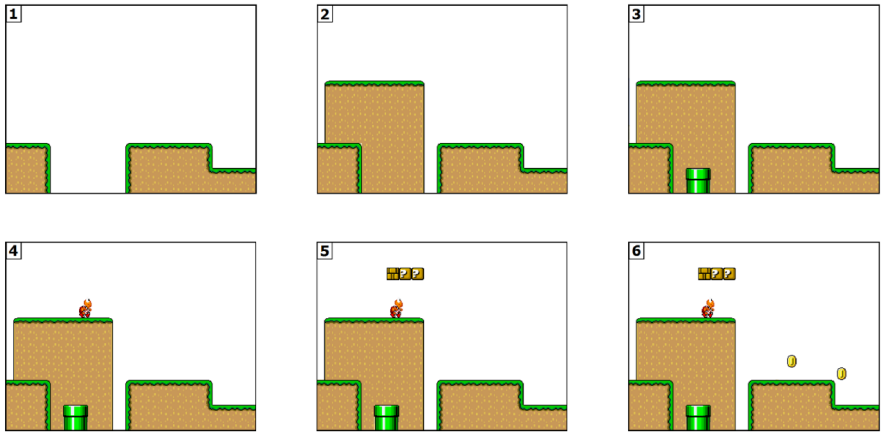


Figura 2.7: Probabilistic Multi-Pass Generator

Al secondo posto si sono classificati Tomoyuki Shimizu e Tomonori Hashiyama. Il loro algoritmo è strutturato in tre moduli separati che interagiscono tra di loro. Il primo modulo (*skill and preference estimator*) analizza le partite svolte dal giocatore e ne compila un profilo che stabilisce quanto è bravo e quali comportamenti tende ad adottare. Il secondo modulo (*parts collector*) raccoglie delle scene generate all'inizializzazione del programma e le cataloga per livello di difficoltà e per tipologia di elementi presenti. Il terzo modulo (*parts connecter*) genera il livello connettendo le scene catalogate dal modulo precedente. Questo algoritmo ha il vantaggio di generare dei livelli che corrispondono alle abilità e alle preferenze del giocatore ma se non si hanno a disposizione un numero sufficientemente

variegato di scene, i livelli generati alla lunga potrebbero risultare monotoni. Questo algoritmo può essere generalmente applicato su un vastissimo numero di contenuti; è sufficiente riadattare i moduli a seconda della tipologia di videogioco con cui si ha a che fare.

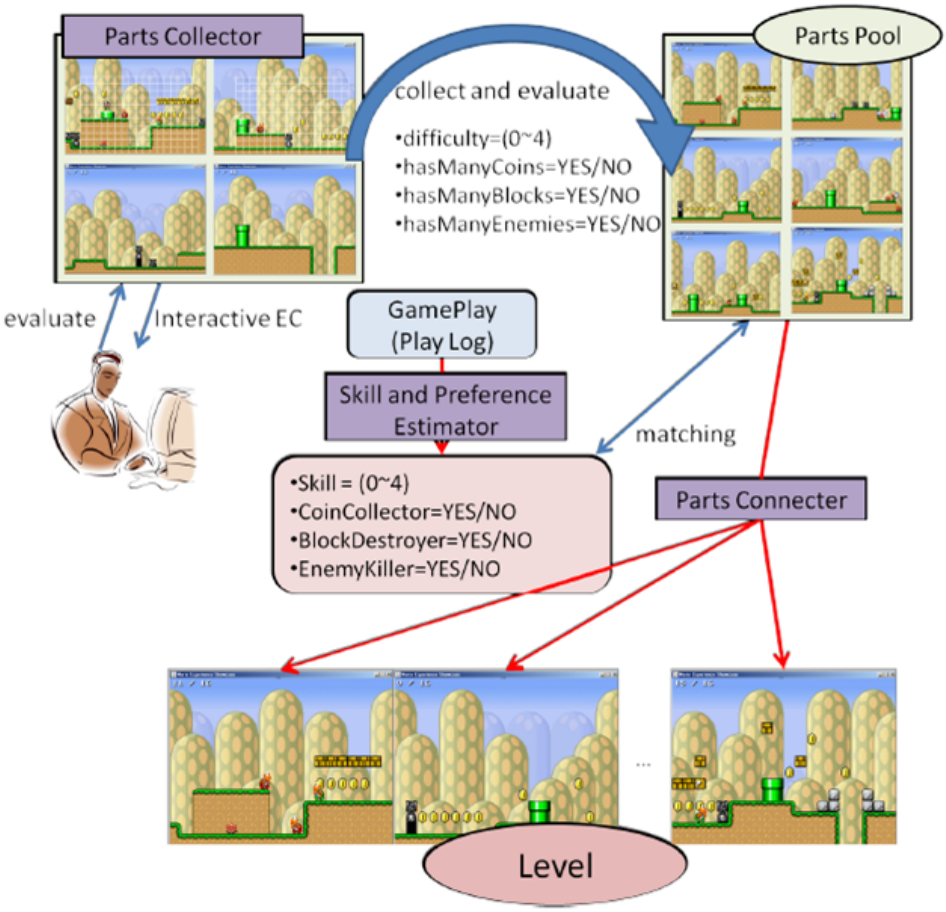


Figura 2.8: Generatore di livelli di Shimizu e Hashiyama

2.5 Sommario

In questo capitolo abbiamo trattato lo stato dell'arte. Nella Sezione 2.1 abbiamo introdotto la generazione procedurale di contenuti nei videogiochi, raccontando la sua storia. Nella Sezione 2.2 abbiamo trattato la classificazione degli algoritmi di generazione procedurale di contenuti. Nella Sezione 2.3 abbiamo analizzato la generazione procedurale di contenuti con algoritmi di ricerca. Nella sottosezione 2.3.1 abbiamo classificato le possibili rappresentazioni dei contenuti candidati. Nella sottosezione 2.3.2 abbiamo trattato delle diverse funzioni di fitness. Nella sottosezione 2.3.3 abbiamo analizzato nello specifico gli algoritmi di ricerca. Nella Sezione 2.4 abbiamo presentato la generazione procedurale di contenuti applicata ai puzzle game. Nella sottosezione 2.4.1 abbiamo analizzato l'esempio del videogioco *Cut The Rope: Play Forever*. Infine nella sottosezione 2.4.2 abbiamo trattato l'esempio del videogioco *Infinite Mario Bros*.

3 Journey at the Edge of the Universe

In questo capitolo viene introdotto *Journey at the Edge of the Universe*, il videogioco da me sviluppato e su cui è incentrata questa tesi. Vengono analizzati nel dettaglio tutti gli elementi di gioco e le funzionalità ad essi associate. In seguito vengono trattati i vari tipi di universo che il giocatore può esplorare. Successivamente viene presentato il sistema di interfaccia con cui l'utente interagisce con il videogioco. Infine vengono elencati tutti gli strumenti utilizzati durante lo sviluppo del videogioco.

3.1 Scopo del gioco

Il videogioco è un puzzle game basato sulla fisica [3] ed è stato sviluppato attorno ad una tematica spaziale. Nel gioco sono state riprodotte delle meccaniche tipiche della fisica come la forza di attrazione tra corpi, la forza propulsiva e il consumo di carburante. Queste meccaniche non sono fisicamente esatte, ma mirano a dare al gioco un'idea di verosimilità.
Il giocatore, interagendo con lo schermo del dispositivo, controlla un razzo e deve guidarlo alla scoperta dell'universo, passando tra vari livelli e interagendo con gli elementi che incontrerà di volta in volta. Il giocatore deve condurre il razzo il più lontano possibile cercando di non esaurire il carburante. Ogni spostamento del razzo richiede un consumo di carburante ed è quindi compito del giocatore cercare di preservarlo e di fare rifornimento quando gli è possibile.
I livelli del videogioco sono generati proceduralmente e questo rende il gameplay potenzialmente infinito.

3.2 Elementi di gioco

In questa sezione vengono presentati il protagonista del videogame e tutti i corpi celesti che andranno a popolare i diversi livelli del gioco.

3.2.1 Razzo

Il razzo è il protagonista del gioco ed è l'unico elemento che il giocatore può controllare. E' un oggetto sempre in movimento e non è possibile farlo fermare. All'inizio del gioco, il razzo ha a disposizione il pieno di carburante (2000 unità) e il giocatore deve essere in grado di gestirlo, calibrando gli spostamenti e cercando di fare rifornimento quando possibile.
Se il razzo entra nel campo di gravitazione di un altro elemento di gioco segue il movimento imposto dalla forza di gravità di quell'elemento. Se invece non subisce nessuna attrazione, continua a muoversi in linea retta. Nel gioco il razzo è schematicamente rappresentato da una sfera di colore rosso.

3.2.2 Pianeta

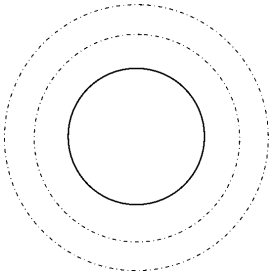


Figura 3.1: Schema di un pianeta

Un pianeta esercita una forza attrattiva verso il suo centro e il razzo può percorrere attorno ad esso delle orbite circolari. Sappiamo che nella realtà le orbite, così come i pianeti, non sono esattamente circolari e la scelta di utilizzare questo tipo di rappresentazione al posto di quella ellittica è stata una scelta semplificativa.
Tutti i pianeti hanno due orbite, una più interna e l'altra più esterna. Il giocatore potrà scegliere di far orbitare il razzo in una di queste due orbite a seconda del proprio bisogno: ad esempio può scegliere di passare dall'orbita esterna a quella più interna per evitare dei satelliti in quella stessa orbita; oppure può scegliere di passare da quella interna a quella più esterna per poter mirare un obiettivo con più semplicità. Questo passaggio da un'orbita all'altra costa al razzo un consumo di carburante, identico per entrambi gli spostamenti. Rimanendo sulla stessa orbita invece il razzo non consuma carburante.

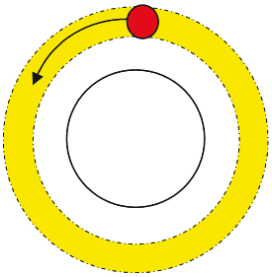


Figura 3.2: Pianeta con razzo in orbita

In Figura 3.2 viene evidenziata l'orbita che il razzo sta percorrendo. La freccia viene mostrata solo a scopo esemplificativo e non viene imposta la rotazione in un solo verso; infatti il razzo potrà orbitare sia in senso orario che in senso antiorario assecondando la traiettoria che ha compiuto per raggiungere il pianeta.

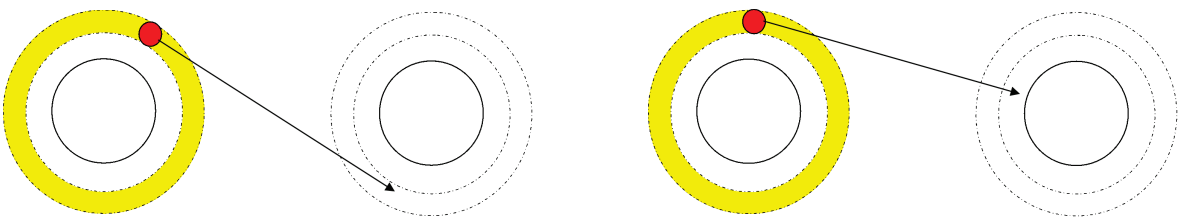


Figura 3.3: Lancio del razzo verso un altro pianeta

In Figura 3.3 viene riportata la traiettoria che compie il razzo quando viene lanciato dall'orbita in cui si sta muovendo. Quando il razzo riceve il comando di lancio, smette di ruotare intorno al pianeta e descrive una traiettoria che è tangente al pianeta nel punto di lancio. Se poi la traiettoria

è orientata verso il centro del pianeta di destinazione, il razzo inizierà a ruotare nell'orbita inferiore. Se invece la traiettoria è sempre orientata verso il pianeta, ma non verso la zona centrale, il razzo ruoterà nell'orbita più esterna.

Esistono diversi tipi di pianeti che permettono una maggiore variabilità nello sviluppo del gioco.

3.2.2.1 Pianeta senza satelliti

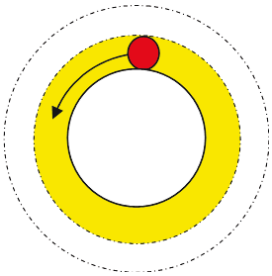


Figura 3.4: Pianeta con razzo nell'orbita inferiore

In nessuna delle due orbite di questo pianeta ci sono dei satelliti e quindi il razzo può orbitare in tutta tranquillità in una qualsiasi delle due orbite.
La scelta dell'orbita spetta al giocatore, tenendo conto che il passaggio da un'orbita all'altra costa del carburante, così come scegliere di lanciare il razzo da un'orbita interna costa più carburante rispetto al lancio da quella più esterna.

3.2.2.2 Pianeta con satelliti

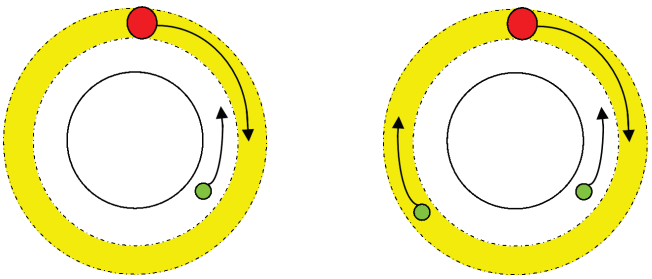


Figura 3.5: Pianeti con satelliti in orbita

Il pianeta con satelliti può avere in orbita uno o al massimo due satelliti. I satelliti sono degli ostacoli che, se colpiti, distruggono il razzo e costringono il giocatore a ricominciare il livello. I satelliti sono schematicamente identificati da sfere di colore verde.
La presenza dei satelliti è comunque limitata al massimo ad uno per orbita. Se il pianeta ha in orbita un solo satellite, il suo verso di rotazione non deve rispettare nessuna particolare regola, mentre se i satelliti sono due, il loro verso di rotazione è sempre opposto. Se il satellite più interno ruota in senso orario, quello esterno ruota in senso antiorario o viceversa.

3.2.2.3 Pianeta esplosivo

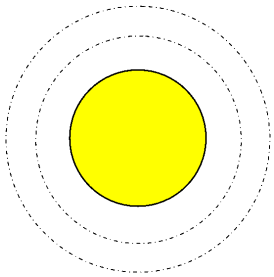


Figura 3.6: Pianeta esplosivo

Il pianeta esplosivo è rappresentato da una sfera di colore giallo e non ha nuovi vincoli sulla presenza o meno di satelliti; starà infatti all'algoritmo che genera il livello scegliere se generare uno, due oppure nessun satellite nelle sue orbite. Il pianeta esplosivo può quindi appartenere alla categoria dei pianeti con satelliti oppure alla categoria dei pianeti senza satelliti.
Quando il razzo entra in una delle sue orbite si innesca un conto alla rovescia di 5 secondi che verrà mostrato sullo schermo del giocatore con una scritta in rosso.
Il giocatore si trova in una situazione in cui deve scegliere il più velocemente possibile come comportarsi; se allo scadere dei 5 secondi il razzo è ancora in orbita intorno al pianeta, questo verrà lanciato via perpendicolarmente, seguendo la traiettoria imposta da una forte esplosione; se invece il razzo avrà lasciato il pianeta prima dello scadere del conto alla rovescia, non gli succederà nulla e potrà continuare a giocare liberamente.
In ogni caso, con l'esplosione, questo pianeta verrà distrutto e non sarà più parte del livello.

3.2.2.4 Pianeta "checkpoint"

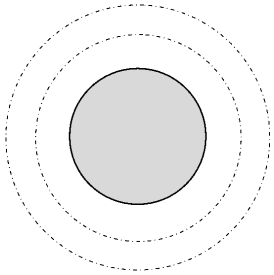


Figura 3.7: Pianeta "checkpoint"

Il pianeta "checkpoint" è un punto di salvataggio. Se il razzo, dopo essere stato in orbita attorno a questo pianeta, dovesse colpire un ostacolo o finire nello spazio più profondo allora il giocatore non sarà costretto a ricominciare il livello dall'inizio, ma ripartirebbe da questo pianeta. Ovvimente, una volta completato il livello, il vecchio pianeta "checkpoint" non agirà più come tale e in caso di distruzione il giocatore riprenderà il gioco dall'inizio del nuovo livello. Come il pianeta esplosivo, anche il pianeta checkpoint può essere un pianeta con satelliti oppure un pianeta senza satelliti. Questo pianeta è identificato da una sfera di colore grigio.

3.2.2.5 Pianeta "fine livello"

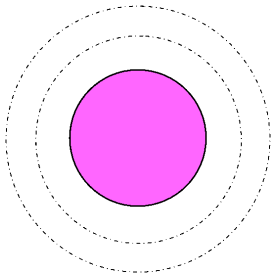


Figura 3.8: Pianeta "fine livello"

Il pianeta "fine livello" rappresenta sia la fine di un livello che l'inizio di uno nuovo. Il giocatore deve esplorare la mappa cercando di raggiungere questo tipo di pianeti (ce ne sarà solo uno per livello) che apriranno le porte ai livelli successivi. Il livello sarà considerato completo quando il razzo sarà rimasto in orbita attorno al pianeta di fine livello per almeno 5 secondi. Se una volta raggiunto questo pianeta il giocatore sceglierà di lanciare il razzo prima del raggiungimento dei 5 secondi oppure se il razzo colpirà un satellite in orbita attorno al pianeta, il livello non sarà considerato completato e sarà necessario ripartire dall'ultimo punto di salvataggio. Per facilitare il giocatore, si è scelto che questo tipo di pianeta potrà avere al massimo un satellite e, se presente, questo sarà collocato nell'orbita più interna. Questo pianeta è rappresentato graficamente da una sfera di colore viola.

3.2.3 Elementi spaziali

I pianeti non sono gli unici corpi celesti che fanno parte del gameplay. Abbiamo aggiunto anche degli altri elementi (alcuni reali, altri no) ma che servono a rendere il gameplay più vario e divertente.

3.2.3.1 Swing

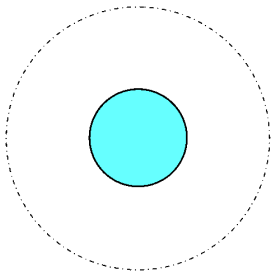


Figura 3.9: Swing

Lo "swing" è un elemento il cui comportamento replica una fionda gravitazionale [14]. Nella realtà la fionda gravitazionale è una tecnica di volo spaziale che utilizza la gravità di un pianeta per alterare il percorso e la velocità di un veicolo spaziale. Nel gioco abbiamo ricreato un elemento che mantiene un comportamento simile. Se il razzo si avvicina allo swing, questo lo attira inizialmente verso di sè e dopo una rotazione di 90 gradi lo allontana modificandone la direzione e la velocità. Questo elemento può essere utilizzato per raggiungere nuovi pianeti oppure per evitare degli ostacoli.
L'area tratteggiata mostrata in Figura 3.9 non è un'orbita come quella dei pianeti incontrati finora, ma un campo di interazione. Se il razzo passa fuori dal campo di interazione, la sua traiettoria non subirà alcuna modifica; in caso contrario interagirà con lo swing e la sua traiettoria risulterà alterata. La fionda gravitazionale devia la traiettoria del razzo di 90° nella direzione verso cui il razzo ruota attorno allo swing.

In Figura 3.10 sono riportati alcuni esempi di interazione tra razzo e swing.

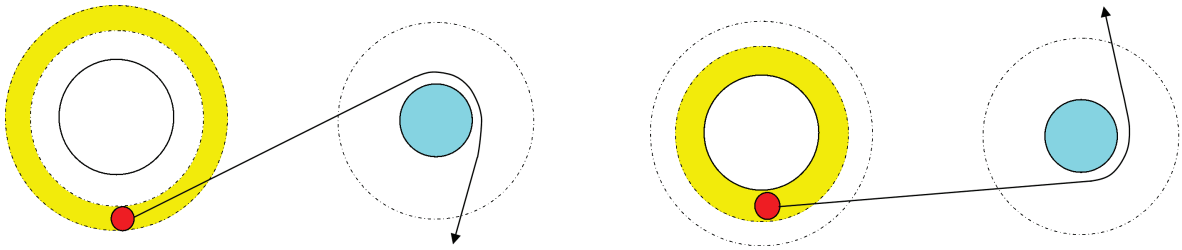


Figura 3.10: Schemi di interazione razzo-swing

3.2.3.2 Wormhole

Figura 3.11: Wormhole

Il wormhole [15] è un noto anche come cunicolo spazio-temporale; è una "scorciatoia" che permetterebbe di viaggiare da un punto all'altro dell'universo ad una velocità superiore rispetto a quella della luce. Di questo elemento si è ipotizzata l'esistenza, ma non è stata ancora empiricamente dimostrata e, se lo fosse, potremmo potenzialmente viaggiare nel tempo. Nel videogame gli wormhole vengono creati sempre in coppia e possono essere usati sia in un senso che nell'altro. Il razzo entra in una estremità del wormhole per poi uscirvi dall'altra. Gli wormhole vengono classificati in cunicoli spazio-temporali intra-universo e cunicoli spazio-temporali inter-universo. Nel videogioco abbiamo scelto di realizzare solo wormhole di tipo intra-universo, quindi la posizione del razzo verrà modificata ma sempre all'interno dello stesso universo. Un'altra scelta implementativa che è stata fatta riguarda il wormhole di uscita: il razzo verrà proiettato esattamente verso destra, indipendentemente dalla direzione con cui era giunto al wormhole di ingresso. Questa scelta è stata presa per semplificare l'uso di questi elementi.
In Figura 3.12 sono riportati due schemi di funzionamento degli wormhole all'interno del gioco.

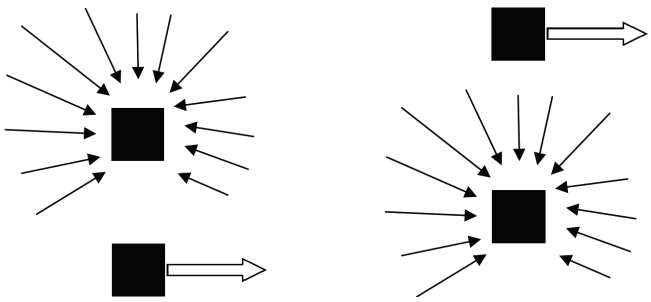


Figura 3.12: Schemi di funzionamento dei wormhole

3.2.3.3 Stazione Spaziale

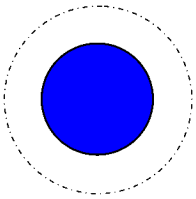


Figura 3.13: Stazione Spaziale

La Stazione Spaziale permette al razzo di fare rifornimento di carburante. Il serbatoio del razzo viene istantaneamente riempito in base a quanto carburante è in grado di fornire la stazione intorno a cui si sta ruotando; esistono infatti stazioni più o meno rifornite. A differenza dei pianeti, la Stazione Spaziale ha una sola orbita disponibile e, in quanto elemento artificiale, non ci possono essere satelliti che gli ruotano intorno.
Fare rifornimento è essenziale per il meccanismo del gioco perchè per compiere ogni movimento il razzo consuma del carburante. La Stazione Spaziale è graficamente rappresentata da una sfera di colore blu.

3.2.4 Ostacoli

Questi elementi sono delle trappole da evitare. Se il razzo colpisce uno di questi oggetti viene immediatamente distrutto e deve ricominciare il livello dall'ultimo punto di salvataggio.

3.2.4.1 Asteroide e Satellite

Questi due elementi hanno la stessa funzione, cioè distruggere tutto ciò che colpiscono. All'interno del gioco, possiamo trovare i satelliti nelle orbite dei pianeti e rappresentano un grosso pericolo per il razzo. Quando il razzo raggiunge un nuovo pianeta, il giocatore deve prestare molta attenzione nel verificare la presenza dei satelliti per evitare di colpirli in fase di ingresso in orbita; ma anche una volta entrati in orbita, il giocatore dovrà fare attenzione ad evitare i satelliti che continuano a ruotare attorno al pianeta.
Gli asteroidi invece non sono legati a nessun elemento del gioco e agiscono come ostacolo per non far seguire al razzo un determinato percorso. Asteroidi e Satelliti sono graficamente rappresentati dallo stesso simbolo, cioè una sfera di colore verde.

3.2.4.2 Buco nero

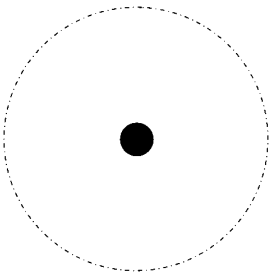


Figura 3.14: Buco nero

Il buco nero [16] è un corpo celeste con una grandissima forza gravitazionale che cerca di catturare tutto ciò che gli passa intorno. Se il razzo entra nel campo di interazione del buco nero, ma in modo molto marginale e quindi abbastanza vicino alla linea dell'orizzonte, la sua velocità gli permetterà di

sfuggire al buco nero e la sua traiettoria risulterà modificata. Se invece il razzo passa troppo vicino al centro del buco nero, questo lo catturerà e lo distruggerà, obbligando il giocatore a ricominciare il livello.
In Figura 3.15 sono riportati due schemi che spiegano il funzionamento di un buco nero all'interno del gioco.

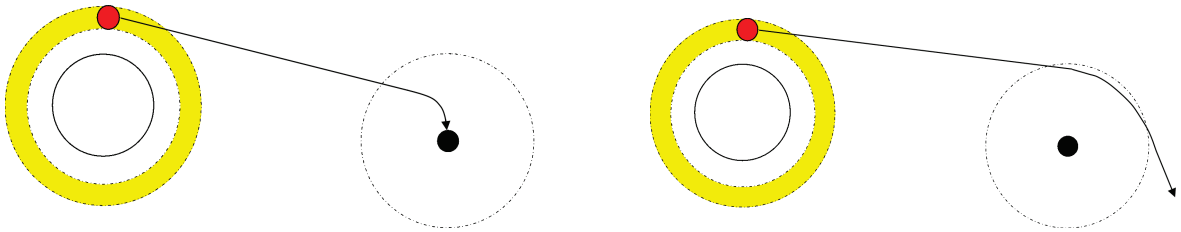


Figura 3.15: Schemi di interazione razzo-buco nero

3.3 Struttura del gioco

Lo spazio di gioco è strutturato in 5 diversi universi in cui il giocatore può affrontare problematiche diverse a seconda di quale universo sceglie di esplorare.

Il primo universo è costituito dal **Tutorial** del gioco. Qui il giocatore incontra ad ogni step uno degli elementi di gioco descritti nella Sezione 3.2. In questo modo il giocatore apprende le caratteristiche di ogni elemento e impara le basi per poter affrontare gli altri universi. Dopo aver presentato i comandi del gioco, e dopo aver introdotto il pianeta "fine livello" e il pianeta senza satelliti, il giocatore è già in grado di provare ad affrontare un primo livello nella modalità più semplice possibile, senza nessun ostacolo. Il giocatore dovrà far muovere il razzo nello spazio di gioco alla ricerca del pianeta di fine livello e, una volta raggiunto questo pianeta, il videogioco genererà automaticamente un nuovo livello accanto al precedente. Grazie alla continua generazione procedurale, il gioco, già soltanto con questi pochi elementi, ha un gameplay potenzialmente infinito. In seguito verrà introdotto il pianeta con i satelliti e anche in questo caso il giocatore verrà invitato a giocare un livello simile al precedente, ma con la difficoltà ulteriore di dover evitare i satelliti che orbitano intorno ai pianeti. Dopo questi elementi fondamentali, al giocatore verranno presentati gli elementi che danno la maggiore variabilità ai livelli dell videogame. Prima il pianeta esplosivo che obbliga il giocatore a decidere in breve tempo quali mosse compiere. In seguito vengono introdotti il Wormhole e lo Swing che permettono al razzo di cambiare traiettoria. Successivamente vengono presentati gli ostacoli da evitare come buchi neri e asteroidi. Infine vengono presentati la Stazione Spaziale, che è l'elemento che permette al razzo di fare rifornimento di carburante, e il pianeta checkpoint che crea un punto di salvataggio da cui il giocatore può ripartire in caso di incidente.
Quelli presentati nel Tutorial sono tutti gli elementi presenti nel gioco quindi, una volta completato il Tutorial, il giocatore ha le basi per poter esplorare uno qualsiasi degli altri universi.

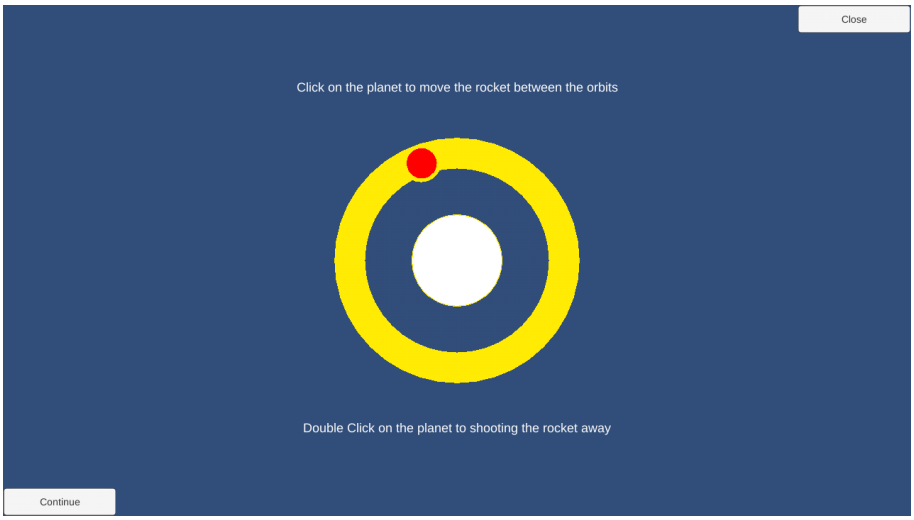


Figura 3.16: I universo: screenshot del Tutorial

Il secondo universo è popolato solo da **pianeti** con satelliti e pianeti senza satelliti. I pianeti di inizio e fine livello possono avere al massimo un satellite, mentre per gli altri pianeti del livello non è stata stabilita una regola. E' quindi compito dell'algoritmo che genera il livello scegliere quanti satelliti generare e dove posizionarli. Un livello con un grande numero di satelliti è generalmente più difficile rispetto a un livello con pochi satelliti. Il giocatore deve guidare il razzo alla scoperta di questo universo, cercando di raggiungere i pianeti posti alla fine di ogni livello. Ogni livello che viene generato ha una difficoltà superiore rispetto a quello appena concluso, infatti i pianeti sono posti, di volta in volta, a distanze sempre maggiori tra loro.

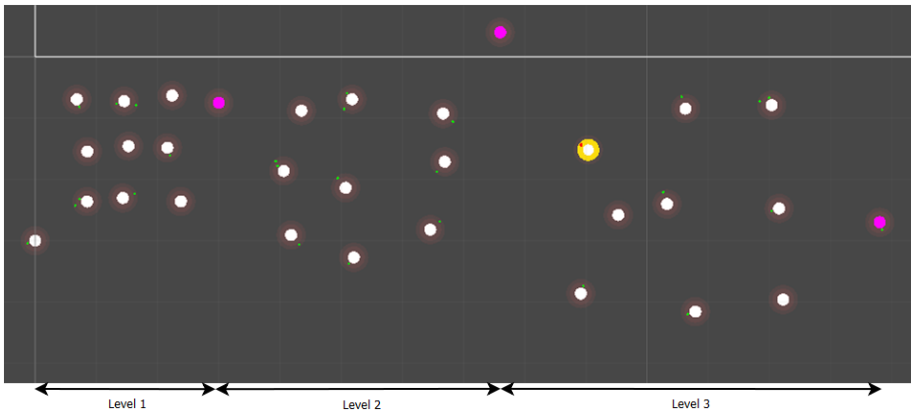


Figura 3.17: II universo: la distanza tra i pianeti aumenta

Il terzo universo è composto da pianeti con satelliti, pianeti senza satelliti e da **asteroidi**. Nel secondo universo l'ambiente di gioco era libero e il razzo poteva muoversi per il livello come meglio credeva, evitando solo i satelliti nelle orbite dei pianeti. In questo universo è invece il livello stesso a decidere il percorso che il razzo dovrà seguire. A seconda della collocazione dei pianeti di inizio e fine livello, l'algoritmo dispone degli asteroidi tra un pianeta e l'altro. Gli asteroidi, se colpiti, distruggono il razzo e costringono il giocatore a ricominciare il livello. Questi elementi possono essere visti sia come un ostacolo, perchè il giocatore deve evitarli per non vedere il razzo distrutto, ma anche come un aiuto. Gli asteroidi infatti delimitano il percorso da seguire e, facendo muovere il razzo in zone libere da asteroidi, si ha la certezza di essere sulla buona strada per completare il livello.

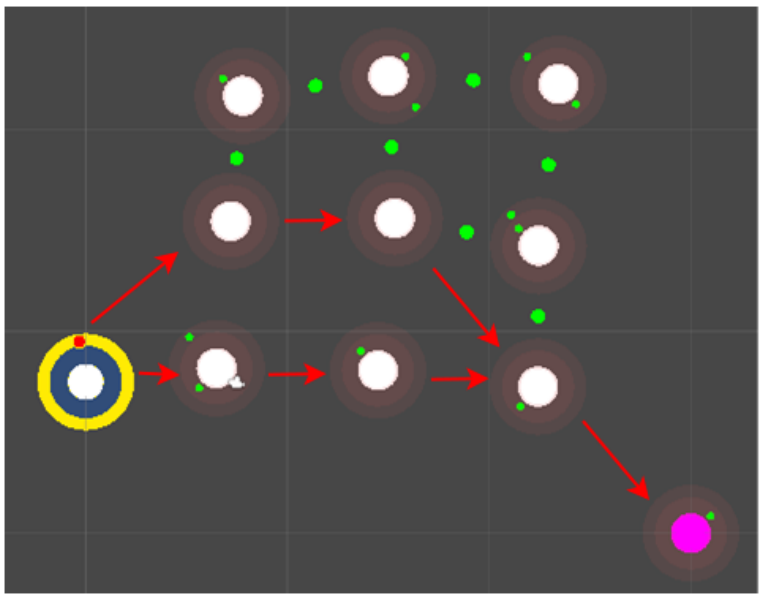


Figura 3.18: III universo: gli asteroidi fanno da ostacolo ma delimitano anche il percorso da seguire

I livelli del quarto universo possono essere popolati da **tutti gli elementi** presentati nella Sezione 3.2, eccezion fatta per gli elementi di inizio e di fine di ogni livello che sono sempre dei pianeti. Ogni elemento ha una certa probabilità di essere presente nel livello. Le probabilità restano immutate al progredire dei livelli esplorati. Gli elementi con la probabilità più alta di essere presenti sono i pianeti perchè sono gli elementi che danno al razzo una maggiore probabilità di sopravvivenza e permettono al giocatore di pianificare il modo in cui affrontare ogni livello. Dopo i pianeti troviamo (in ordine decrescente di probabilità) le Stazioni Spaziali, Swing, Wormhole e Buchi neri.

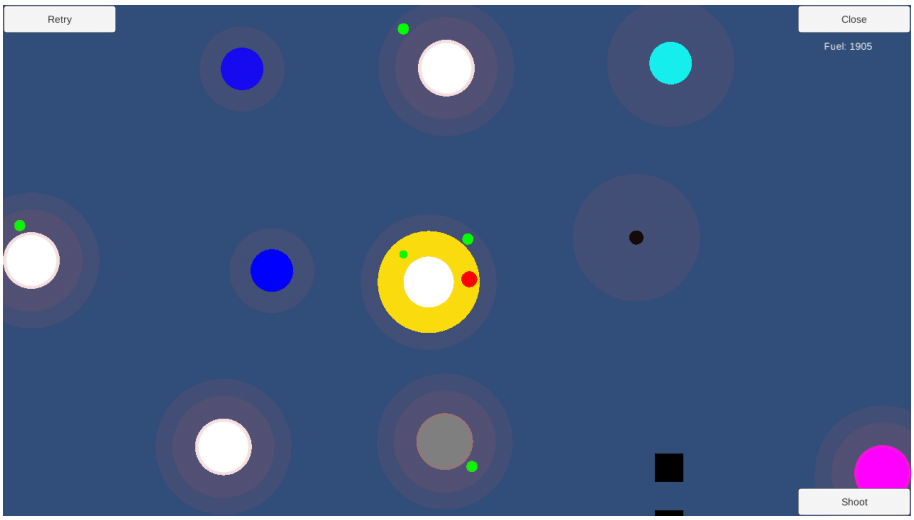


Figura 3.19: IV universo: screenshot di un livello popolato da diversi tipi di elementi

Il quinto universo è costituito da livelli formati esclusivamente da pianeti con satelliti e pianeti senza satelliti. Questo universo si differenzia dal secondo universo per il modo in cui i pianeti vengono posizionati all'interno del livello. In questi livelli le **posizioni** dei pianeti **non** sono più **vincolate** da uno schema preciso ma hanno una maggiore variabilità e danno al giocatore un maggiore senso di esplorazione. Questo universo risulta essere piuttosto diverso da quelli precedenti e il giocatore potrebbe trovare una maggiore difficoltà nell'affrontarlo.

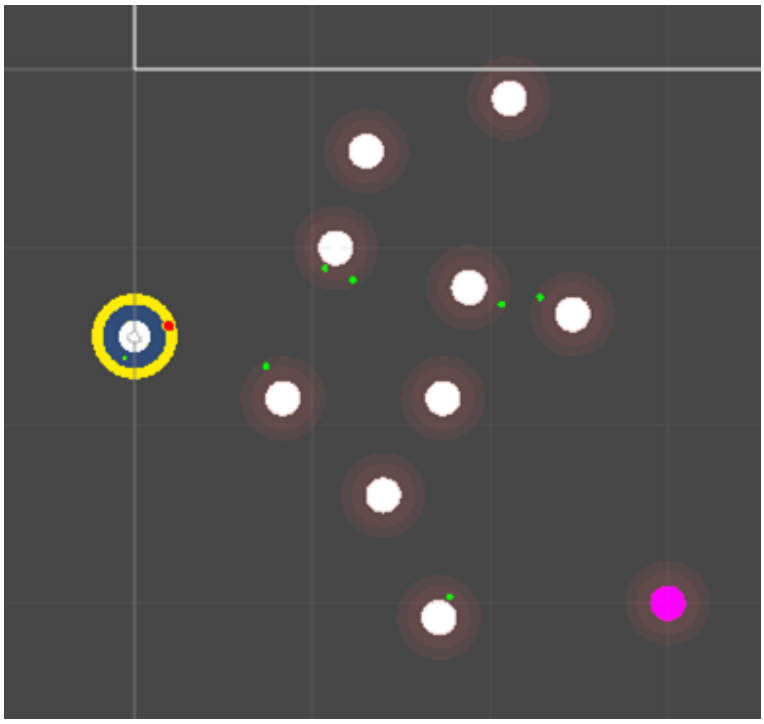


Figura 3.20: V universo: i pianeti hanno una diversa disposizione

3.4 Interfaccia grafica e sistema di input

Il videogioco è stato sviluppato per dispositivi mobile, ma è anche possibile giocarne una versione per computer. Nella versione per computer il giocatore può interagire con il videogioco cliccando con il mouse, mentre nella versione mobile il giocatore deve toccare lo schermo. Di seguito tratteremo l'azione del *tap* (il tocco con le dita sullo schermo touchscreen) come se fosse il *click* del mouse. Queste azioni sono diverse solo perchè vengono effettuate su dispositivi che devono essere controllati in maniera differente, ma gli effetti che producono sul gameplay sono gli stessi.
Quando il razzo sta orbitando intorno ad un pianeta, il giocatore può scegliere di far spostare il razzo da un'orbita all'altra. Questa azione può essere compiuta cliccando una sola volta sul pianeta attorno a cui il razzo sta orbitando. Dopo aver controllato che il razzo abbia nel serbatoio carburante a sufficienza per compiere questo spostamento, la sua orbita verrà modificata da interna ad esterna o viceversa. Se invece il razzo non ha abbastanza carburante per spostarsi verrà riposizionato all'inizio del livello con il pieno di carburante.
Inoltre il giocatore può lanciare il razzo da un pianeta verso un altro corpo celeste. Questa azione viene eseguita facendo un doppio click sul pianeta attorno a cui il razzo sta orbitando. In alternativa il giocatore può premere un apposito bottone che appare sullo schermo.

All'avvio il videogioco presenta un menù iniziale che permette al giocatore di scegliere quale dei diversi universi esplorare. Si accede ad ogni universo cliccando su uno dei bottoni mostrati sullo schermo. Il bottone "*PCG_ Tutorial*" proietta il giocatore in un universo in cui ha la possibilità di imparare i più basilari meccanismi del gioco e di interagire con tutti i corpi celesti che incontrerà nei vari universi. Il bottone "*PCG_ Continue*" porta il giocatore nel secondo universo, i cui livelli sono composti solo da pianeti e attorno ai quali possono ruotare dei satelliti. Il bottone "*PCG_ Obstacle*" porta il giocatore nel terzo universo in cui i livelli sono popolati solo da pianeti con o senza satelliti e da asteroidi. Gli asteroidi servono ad aumentare le difficoltà di ogni livello e al tempo stesso a guidare il giocatore verso la fine del livello stesso. Il bottone "*PCG_ RandomObject*" conduce il giocatore nel quarto universo, dove potrà incontrare in ogni livello tutti gli elementi di gioco presentati nella

Sezione 3.2. Infine il bottone "*PCG_ RandomPlace*" porta il giocatore nel quinto universo popolato solo da pianeti con o senza satelliti, ma le cui posizioni all'interno di ogni livello sono molto diverse rispetto a quelle dei corpi celesti che popolano tutti gli altri universi.
Sulla schermata principale è anche presente un bottone "*Quit*" che permette al giocatore di chiudere l'applicazione e terminare il gioco.

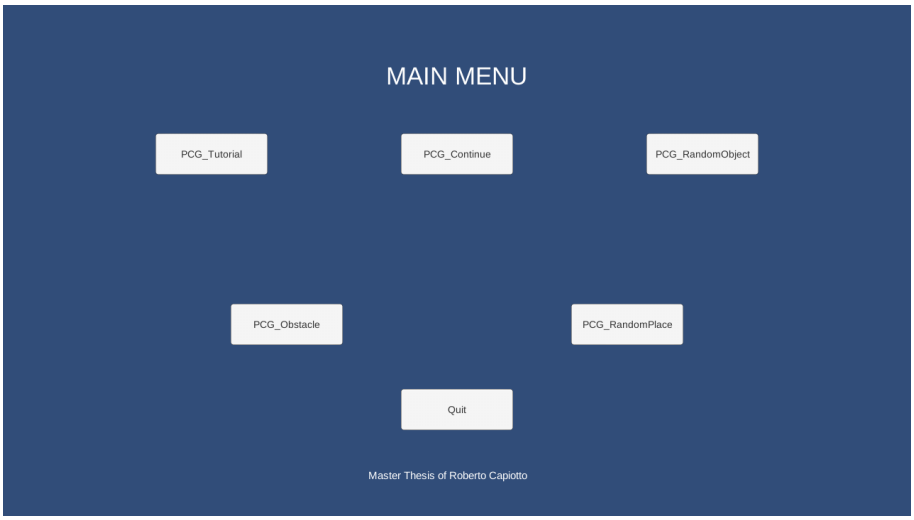


Figura 3.21: Schermata Home

Quando il giocatore entra in uno degli universi vedrà negli angoli della schermata di gioco dei bottoni che gli permettono di compiere diverse operazioni.
Nell'angolo in basso a sinistra viene mostrato il bottone "**Continue**" che permette al giocatore di passare allo step successivo del Tutorial. Nell'angolo in basso a destra è visibile il bottone "**Shoot**". La pressione di questo bottone permette di lanciare il razzo e può sostituire il doppio click sul pianeta attorno a cui il razzo sta orbitando. Nell'angolo in alto a destra è presente il bottone "**Close**" che chiude l'intero universo in cui si sta giocando per riportare il giocatore alla schermata principale del gioco. Infine è collocato nell'angolo in alto a sinistra il bottone "**Retry**" che cancella il livello che il giocatore sta affrontando per generarne uno nuovo e diverso dal precedente. E' importante soffermarsi sull'uso di quest'ultimo bottone perchè fornisce al giocatore la possibilità di sostituire dei livelli generati proceduralmente che ritiene troppo difficili da affrontare. In questo modo il giocatore, trovandosi in una situazione difficoltosa, viene spinto a continuare a giocare affrontando livelli con caratteristiche a lui più congeniali.
Il bottone "Continue" è presente solo nel Tutorial. Il bottone "Retry" è presente in tutti gli universi di gioco e nelle schermate del Tutorial dove è presente un intero livello di gioco. I bottoni "Shoot" e "Close" invece sono sempre presenti durante il gameplay.
Sullo schermata di gioco è anche presente un indicatore che ci informa in ogni momento su quanto carburante è presente nel serbatoio del razzo.

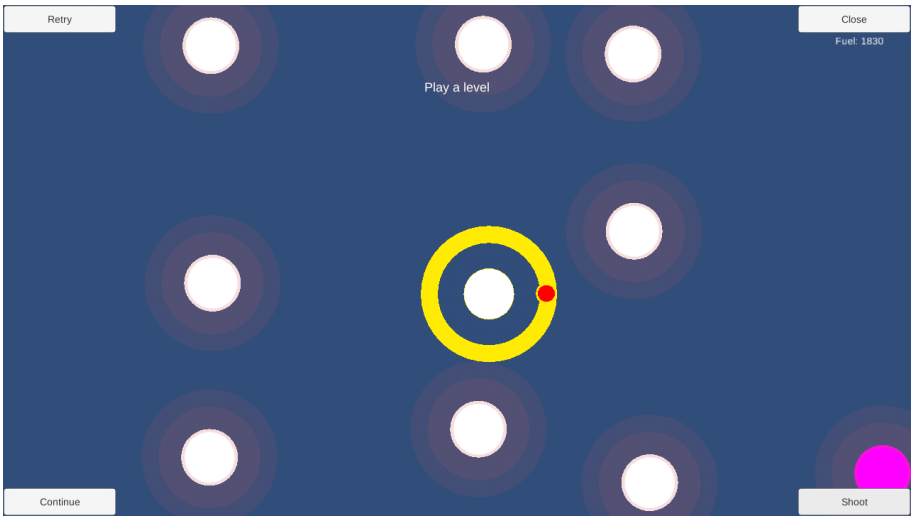


Figura 3.22: Screenshot della schermata di gioco con tutti i bottoni

3.5 Strumenti utilizzati

Il computer che ho utilizzato per questo lavoro ha un sistema operativo **Windows** 7 Professional su cui sono stati installati i software descritti in seguito.
Unity 4.6.7f1 [17] è l'ambiente di sviluppo utilizzato per realizzare il videogioco. Unity è in grado di gestire il rendering grafico degli elementi di gioco, ricreare gli effetti di luce e simulare le meccaniche fisiche. Per la stesura degli script applicati agli elementi di gioco è stato usato **MonoDevelop** 4.0.1 [18] e il linguaggio usato è il C#. Lo strumento utilizzato per conservare il codice del progetto è stato **GitHub** 3.0.7.1 [19] che permette di avere a propria disposizione tutte le versioni dei codici caricati, con le relative modifiche effettuate volta per volta. La versione free di Unity permette di sviluppare per moltissime piattaforme, ma io ho utilizzato solo la piattaforma Windows e quella Android. *Journey at the Edge of the Universe* è stato installato sul mio **Samsung Galaxy S3** che ha un sistema operativo Android 4.3 (API 18). Per rendere possibile lo sviluppo sui dispositivi Android è stato necessario installare **Android SDK** 24.3.3 [20], aggiornata ad Android 6.0 (API 23) e **Java Development Kit** (JDK 8u66) [21].

L'intero codice del progetto è disponibile online al seguente indirizzo:
`https://github.com/roberto-capiotto/MasterThesis/`

3.6 Sommario

In questo capitolo abbiamo introdotto *Journey at the Edge of the Universe*. Nella Sezione 3.1 abbiamo presentato la tematica del gioco e l'obiettivo che il giocatore deve raggiungere. Nella Sezione 3.2 abbiamo analizzato tutti gli elementi di gioco. Nella sottosezione 3.2.1 abbiamo introdotto il razzo, protagonista del videogioco e unico elemento che il giocatore può controllare. Nella sottosezione 3.2.2 abbiamo elencato i vari tipi di pianeti che popolano i vari universi di gioco. Nella sottosezione 3.2.3 abbiamo presentato altri elementi spaziali diversi dai pianeti. Nella sottosezione 3.2.4 abbiamo introdotto gli ostacoli. Nella Sezione 3.3 abbiamo trattato la struttura del gioco e la sua suddivisione nei vari universi. Nella Sezione 3.4 abbiamo analizzato l'interfaccia grafica e il sistema di input del videogioco. Nella Sezione 3.5 abbiamo elencato gli strumenti utilizzati durante lo sviluppo del videogioco.

4 Algoritmi

In questo capitolo analizziamo gli algoritmi che generano i livelli di ogni universo di *Journey at the Edge of the Universe* e illustriamo la loro implementazione. In seguito viene effettuato un confronto tra la teoria della generazione procedurale di contenuti e quanto invece è stato realizzato nella pratica.

4.1 Costruzione dei livelli

La costruzione dei livelli avviene utilizzando un diverso algoritmo per ogni universo di gioco. Il razzo deve sempre rimanere visibile a video, altrimenti il giocatore non potrebbe controllarlo a dovere e non potrebbe valutare con correttezza quando è il momento di compiere una determinata operazione. Esiste poi la possibilità di muoversi avanti e indietro tra i livelli disponibili dello stesso universo; il videogioco, quando genera un nuovo livello, non cancella il livello appena concluso. Il giocatore è quindi libero di esplorare questi livelli sia verso destra, alla scoperta dell'ignoto, sia verso sinistra, ripercorrendo un cammino già compiuto.

Il **Tutorial** è prevalentemente statico perchè è strutturato in modo tale da permettere al giocatore di imparare le basi per affrontare tutti gli universi di gioco. L'algoritmo che genera proceduralmente dei livelli all'interno del tutorial viene eseguito una prima volta dopo che il giocatore è riuscito a far muovere il razzo tra i pianeti senza satelliti e poi una seconda volta quando il giocatore guida il razzo tra i pianeti con satelliti. Questo algoritmo è pressochè identico a quello che genera i livelli del secondo universo, eccezion fatta per la prima esecuzione in cui i pianeti non hanno nessun satellite in orbita.

I livelli del Tutorial e quelli del **secondo universo** si sviluppano all'interno di una immaginaria griglia di gioco di dimensioni 5x5, mostrata in Figura 4.1. A video viene mostrata solo una porzione della corrente scena di gioco, un 3x3 delimitato dalla linea tratteggiata. In questa immaginaria griglia 5x5, il pianeta di partenza viene collocato nella colonna di sinistra. Il posizionamento del pianeta in una delle cinque righe disponibili è stabilito da una funzione casuale che restituisce l'indice della riga in cui il pianeta deve essere posizionato.
Nel centro della griglia vengono collocati gli altri pianeti che fanno da raccordo tra l'inizio e la fine del livello. Questi pianeti vengono disposti in maniera tale da dare ogni volta al giocatore un livello differente. I pianeti vengono generati seguendo l'ordinamento per colonne e la posizione di ogni pianeta è calcolata da una funzione che tiene conto delle coordinate di questo pianeta relativamente alla griglia e del numero progressivo di livelli completati in questo universo, più una componente casuale che ne varia le coordinate ogni volta che l'algoritmo viene eseguito. La funzione è calibrata in modo tale da far rimanere ogni pianeta all'interno della sua cella, altrimenti si rischierebbe di fare sovrapporre due pianeti e in un caso come questo, il razzo avrebbe un comportamento non prevedibile e non gestibile.
Infine, dopo aver generato tutti i pianeti del livello, si crea il pianeta "fine livello" che viene collocato nell'ultima colonna della griglia di gioco. A stabilire la riga in cui viene posizionato è una funzione casuale che ci permette di avere livelli con il pianeta finale in posizione molto variabile e che danno al giocatore il senso dell'esplorazione.
Quando si genera ogni pianeta, insieme ad esso vengono generati anche i satelliti che gli orbitano intorno. I pianeti di inizio e fine livello possono contenere al più un satellite e, se presente, questo sarà posizionato nell'orbita più interna. Questa scelta è dovuta a una semplificazione del gioco; in

questo modo il giocatore ha la possibilità di completare il livello con maggiore facilità e di restare in orbita per più tempo su questi pianeti per pianificare la strategia per il prossimo livello da affrontare.
A stabilire la presenza dei satelliti intorno ad ogni pianeta è una funzione casuale che restituisce una gamma di valori che vengono semanticamente associati a "nessun satellite", "satellite nell'orbita esterna", "satellite nell'orbita interna" e "satelliti in entrambe le orbite". Più alto sarà il numero di satelliti generati e maggiore sarà la difficoltà del livello.
Dopo aver completato un livello, il videogioco ne genera immediatamente uno successivo che si differenzia dal precedente in un paio di caratteristiche. Innanzitutto il pianeta "fine livello" del livello appena concluso, diventa il nuovo pianeta di inizio livello: il razzo rimarrà in orbita intorno allo stesso pianeta e la telecamera scorrerà verso destra rivelando un nuovo livello inesplorato. Inoltre i pianeti del nuovo livello sono più distanti tra loro rispetto a quelli del livello precedente e questo fattore rende più difficile il gioco perchè il giocatore ha bisogno di una maggiore precisione per lanciare il razzo da un pianeta all'altro.

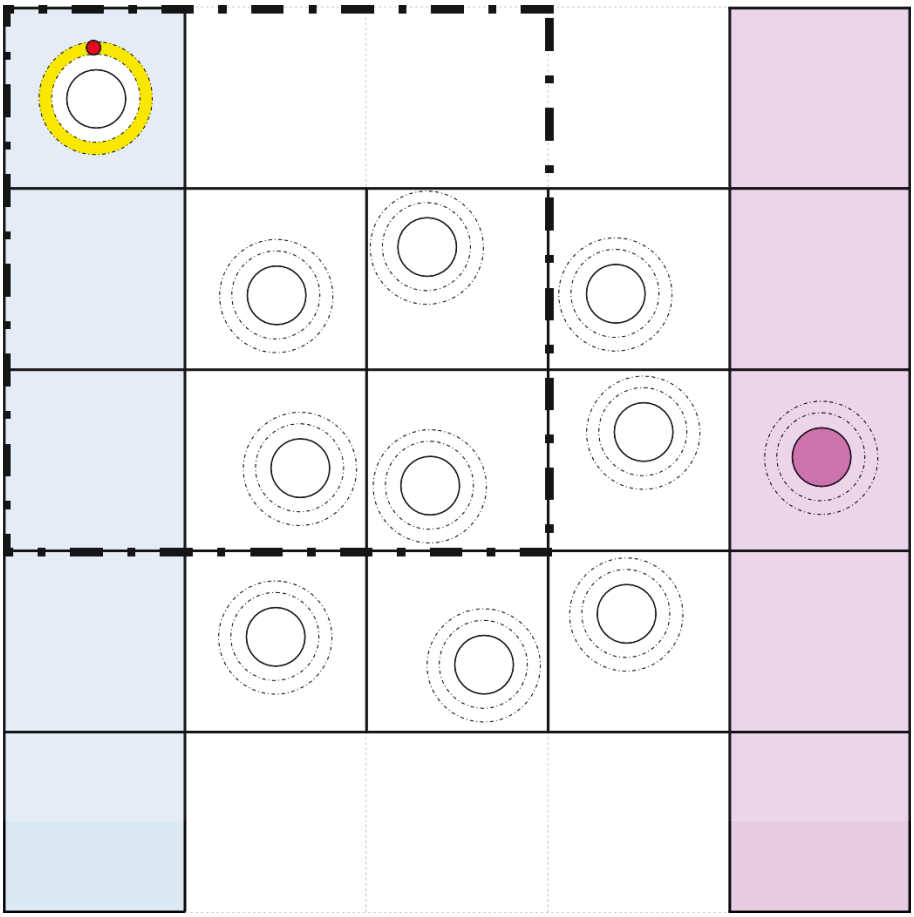


Figura 4.1: Schema di un livello del II universo

I livelli del **terzo universo** ricalcano, per quanto riguarda il posizionamento dei pianeti, quelli del secondo universo.
L'algoritmo che genera questi livelli è lo stesso usato per i livelli del secondo universo, ma dopo aver generato i pianeti del livello passa a una seconda fase. L'algoritmo valuta la posizione nella griglia dei pianeti di inizio e fine livello e sulla base del loro posizionamento calcola i percorsi più rapidi per completare il livello. Lungo questi percorsi l'algoritmo non mette nessun ostacolo, mentre lungo i percorsi più lunghi posiziona degli asteroidi. Questi asteroidi vengono posizionati nel punto medio tra i due pianeti che il razzo non deve raggiungere.
Al completamento del livello, l'algoritmo genera un nuovo livello e il vecchio pianeta di fine livello assume il ruolo di nuovo pianeta di inizio livello.

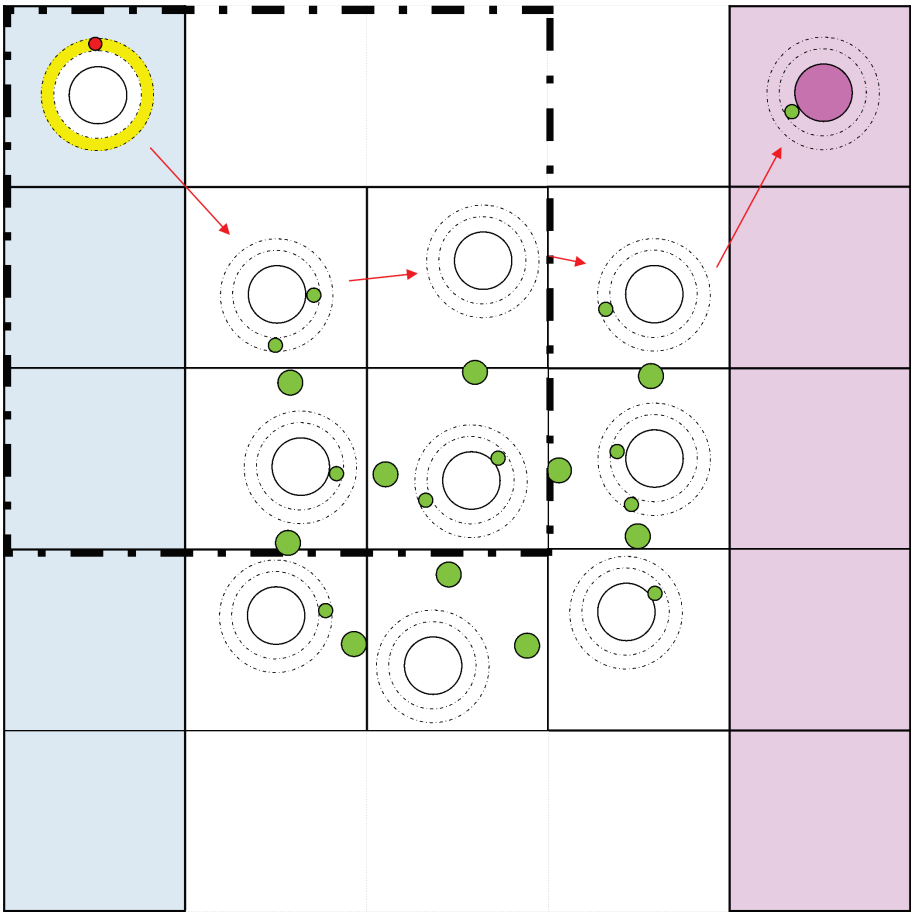


Figura 4.2: Schema di un livello del III universo

Anche i livelli del **quarto universo** sono basati su uno schema a griglia, ma stavolta gli oggetti che sono generati non sono più soltanto dei pianeti.
L'elemento di inizio livello è un pianeta e, come negli altri universi, può non avere satelliti in orbita oppure averne uno solo nell'orbita inferiore. La presenza del satellite, come in precedenza è stabilita da una funzione casuale. I 9 elementi su cui si sviluppa il gioco possono essere tutti gli elementi presentati nella Sezione 3.2 l'algoritmo che li genera si basa su un calcolo probabilistico.

Elemento	Probabilità di generazione
Pianeta con satelliti	50 %
Pianeta checkpoint	10 %
Pianeta esplosivo	10 %
Swing	10 %
Stazione Spaziale	10 %
Buco Nero	5 %
Wormhole	5 %

Tabella 4.1: Tabella probabilistica sulla generazione degli elementi

L'elemento che ha di gran lunga la maggiore probabilità di essere generato è il pianeta con satelliti. E' stata presa questa decisione perchè per il giocatore è l'elemento più semplice da gestire. Dopo aver scelto quale elemento deve essere generato, l'algoritmo si occupa di calcolarne la sua posizione tramite una funzione che basa la sua scelta sulla posizione che questo elemento assume nella griglia e sul numero di livelli completati in questo universo, più una componente casuale che varia la posizione

degli oggetti ogni volta che l'algoritmo viene eseguito. Infine l'elemento di fine livello è ancora un pianeta con un satellite nell'orbita inferiore oppure senza satelliti.
Quando il livello viene completato, il vecchio pianeta di fine livello diventa il nuovo pianeta di inizio livello e l'algoritmo viene nuovamente eseguito per generare un nuovo livello con le stesse stime di probabilità ma con distanze maggiori tra i vari elementi.

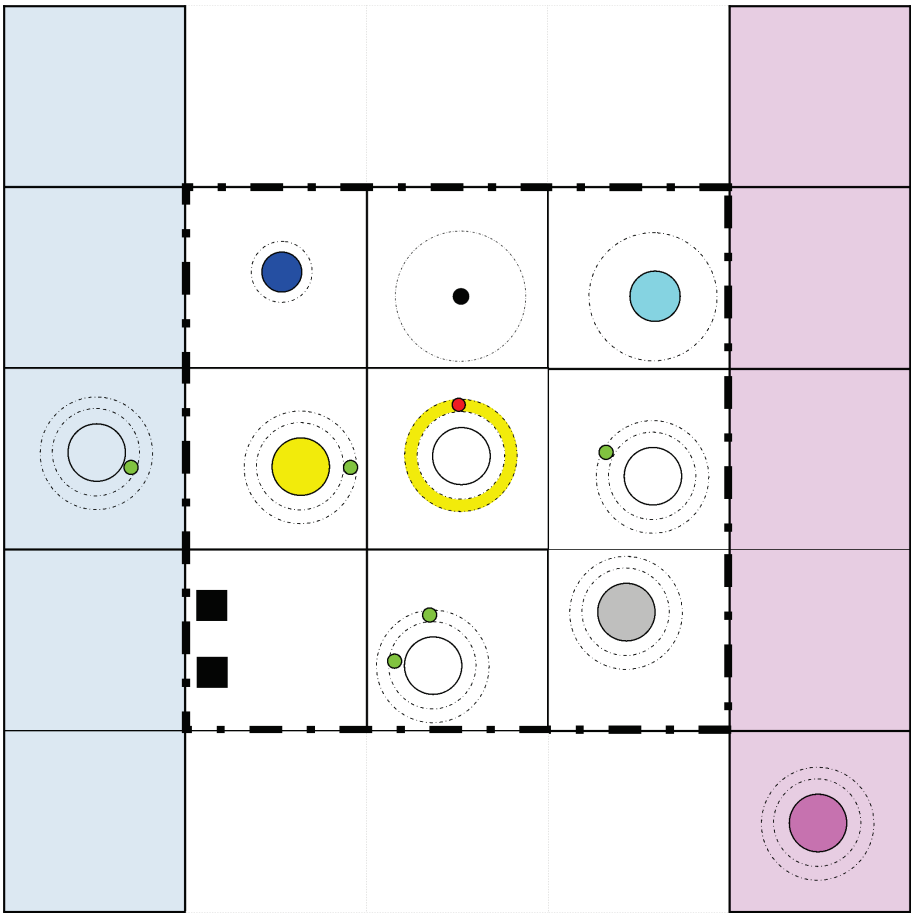


Figura 4.3: Schema di un livello del IV universo

I livelli del **quinto universo**, invece, sono generati con un algoritmo totalmente differente dai precedenti. Tutti gli elementi di questo universo sono pianeti con satelliti. Ogni livello ha i due pianeti di partenza e di arrivo collocati in posizioni casuali, ma a una distanza fissa tra di loro. Una volta scelta la posizione del pianeta di partenza, l'algoritmo che genera il livello cerca di posizionare i pianeti ad una distanza superiore a quella minima, per non sovrapporre i due pianeti, ma inferiore rispetto a quella a cui è posizionato il pianeta di "fine livello". Oltre alla distanza è importante considerare anche la variazione in termini angolari che viene applicata al posizionamento del pianeta. In pratica vengono generati due parametri (ϱ e ϑ) che rappresentano rispettivamente la distanza a cui deve essere posto il nuovo pianeta rispetto a quello di inizio livello e l'angolo che il nuovo pianeta deve idealmente formare con il pianeta di partenza. Se la coordinata scelta è libera, si genera un pianeta in quella posizione, si valuta la generazione dei satelliti nelle sue orbite e poi si passa a cercare di posizionarne uno nuovo. Invece se generare il pianeta nella coordinata scelta andrebbe a sovrapporlo con un altro pianeta già presente nel livello, la posizione scelta non è adatta e sarà necessario generarne una nuova. Questa operazione viene ripetuta finchè tutti i pianeti del livello non sono stati posizionati in maniera corretta.
Una volta completato il livello, il gioco ne genera uno nuovo facendo diventare il precedente pianeta di fine livello il nuovo pianeta di inizio livello. Con il progredire dei livelli viene aumentata la distanza da percorrere tra l'inizio e la fine del livello e di conseguenza viene aumentata anche la distanza a

cui si possono trovare i vari pianeti che ne compongono il percorso: questo rende i livelli sempre più difficili. Il range di accettabilità per il parametro ρ aumenta sempre in ogni livello, mentre il range per il parametro ϑ rimane bloccato tra $\pm$ 1 radiante.
In questo universo, sullo schermo di gioco viene sempre mostrato il razzo in posizione centrale. Questa modalità di ripresa permette al giocatore di vedere cosa ha nelle vicinanze, ma lo obbliga anche a spostarsi verso delle zone inesplorate, alla ricerca del pianeta di "fine livello".

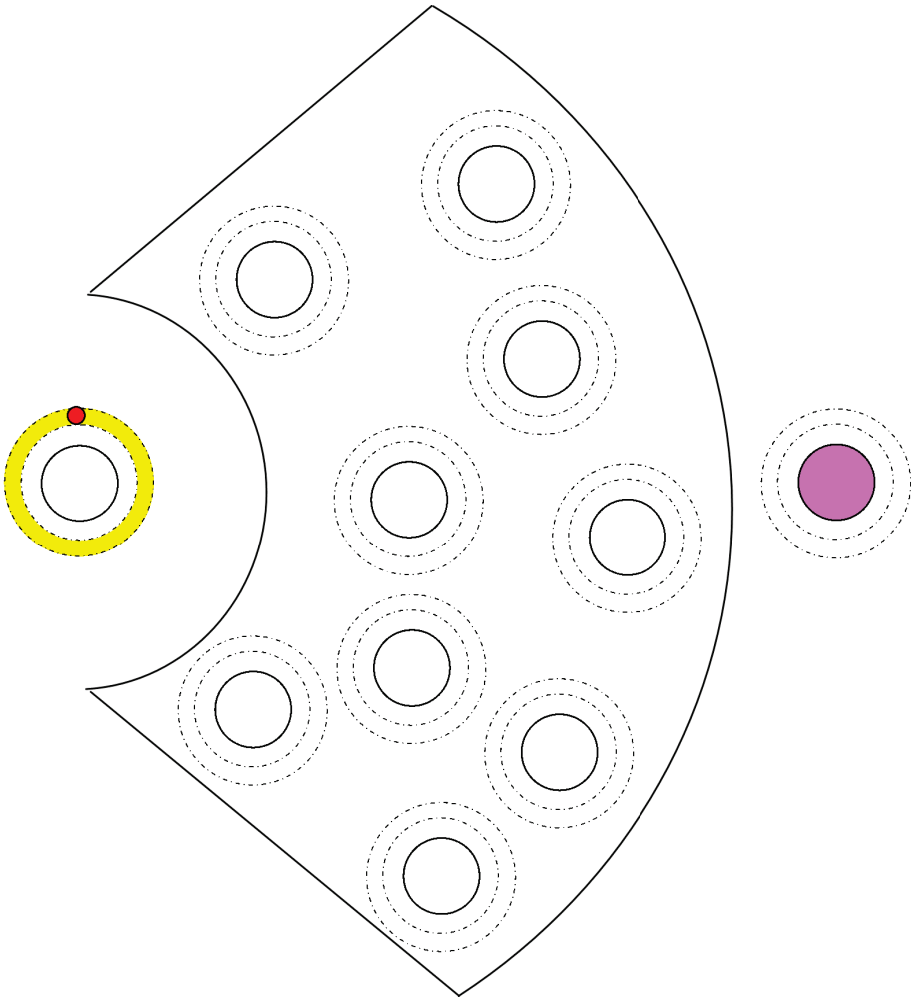


Figura 4.4: Schema di un livello del V universo

4.2 Confronto tra la teoria della PCG e la pratica

Per fare maggiore chiarezza sugli algoritmi usati è utile fare un confronto tra quanto trattato nella teoria generale della generazione procedurale di contenuti (vedi Sezione 2.2) e quanto poi è stato applicato nella realizzazione del videogioco (vedi Sezione 4.1).

4.2.1 Classificazione degli algoritmi

L'algoritmo di generazione dei contenuti impiegato nel videogioco è di tipo **online**, infatti solo quando si raggiunge un determinato pianeta il gioco genera immediatamente un nuovo livello giocabile dove prima non c'era nulla. Avere un algoritmo di tipo offline sarebbe molto oneroso in quanto richiederebbe di generare tutti i livelli prima di iniziare il gioco, senza sapere effettivamente quanti di essi verranno affrontati dal giocatore.

Journey at the Edge of the Universe è pensato per non avere un percorso predefinito da seguire; l'unico pianeta che possiamo considerare come elemento **necessario** è quello di fine livello mentre tutti gli altri elementi generati proceduralmente possono essere considerati come **opzionali**. Il pianeta di fine livello è necessario perchè è la chiave per passare di livello in livello e senza di esso il gioco non avrebbe uno scopo chiaro da portare a termine. Gli altri elementi, in particolare asteroidi e buchi neri, sono elementi addizionali ma che danno al videogioco una aggiunta in termini di gameplay.

L'algoritmo utilizzato per generare gli elementi dei livelli di tutti gli universi è di tipo **parameter vector**. Esiste infatti un vettore con dei parametri che specifica delle regole a cui attenersi per una corretta generazione del livello. Utilizzare un algoritmo di tipo **random seeds** significherebbe basare interamente la generazione su dei numeri casuali. Questo sarebbe possibile ma rischieremmo di avere dei livelli molto complicati da giocare e che richiederebbero molto tempo per verificare la loro giocabilità.

L'algoritmo di PCG usato in *Journey at the Edge of the Universe* è di tipo **stocastico**; gli elementi generati per ogni livello hanno una grande variabilità, sia a livello di posizione che a livello di tipologia stessa dell'elemento che sarà generato. Fare uso di un algoritmo deterministico in questo contesto genererebbe sempre lo stesso livello, perfettamente giocabile, ma il giocatore perderebbe tutta la componente di gioco legata all'esplorazione.

Journey at the Edge of the Universe fa uso di sia di algoritmi constructive che di algoritmi generate-and-test. Gli algoritmi applicati nei primi quattro universi sono di tipo **constructive**; i contenuti vengono generati una sola volta e abbiamo la certezza che questi siano corretti. L'algoritmo **generate-and-test** invece viene utilizzato nel quinto universo; si fa un controllo "a monte", si elegge un candidato e si verifica se esso è adatto al livello che si sta generando, evitando quindi che due pianeti vengano messi in posizioni sovrapposte.

4.2.2 Rappresentazione dei contenuti

Il genotipo è un vettore di parametri che permette di codificare il contenuto e può essere diverso in ognuno degli universi di gioco, come mostrato nella tabella.

Universo	Elenco parametri
I, II e III universo	Coordinate di inizio del livello Posizione dell'elemento nella griglia Numero di livelli completati Numero dei satelliti in orbita a ciascun pianeta Componente casuale per l'offset
IV universo	Coordinate di inizio del livello Posizione dell'elemento nella griglia Numero di livelli completati Tipologia di elemento da generare Numero dei satelliti in orbita (se attinente) Componente casuale per l'offset
V universo	Coordinate del pianeta di inizio livello Numero dei livelli completati Posizioni dei pianeti già generati ρ : distanza dal pianeta di inizio livello ϑ : variazione angolare rispetto al pianeta di inizio livello Numero dei satelliti in orbita a ciascun pianeta

Tabella 4.2: Parametri che identificano il genotipo dei contenuti

Il fenotipo è invece rappresentato dall'elenco delle coordinate che avranno gli elementi di gioco nel livello che viene generato. La codifica che lega genotipo e fenotipo è diretta: bastano infatti pochi calcoli per risalire da una rappresentazione all'altra. Tutti questi parametri assumono valori reali e l'algoritmo che li analizza può svolgere il suo compito in maniera più rapida, avvalendosi della dimensionalità e della località delle informazioni. La dimensionalità è rispettata, anzi qualche parametro in più permetterebbe di avere dei livelli ancora più interessanti, mantenendo il tempo di esecuzione dell'algoritmo ad un livello più che accettabile. Anche la località è rispettata infatti, aumentando o diminuendo di poco il numero dei satelliti o le coordinate degli elementi, il fenotipo e i valori di fitness subirebbero variazioni minime.

4.2.3 Funzione di fitness

Nei primi quattro universi, la funzione di fitness accetta sempre tutti i contenuti candidati perchè la loro generazione è codificata in maniera tale da creare sempre dei livelli giocabili.
Nel quinto universo, invece, il videogioco utilizza una funzione di fitness diretta ma con un range di valori limitato. Generalmente i valori di fitness sono reali e la loro scelta dipende dal superamento o meno di una determinata soglia. Nel nostro caso invece la funzione restituisce solo un valore di tipo {0,1} dove il valore zero indica che il candidato non è accettabile e va rigenerato, mentre il valore uno indica che il candidato è idoneo.

4.3 Sommario

In questo capitolo abbiamo introdotto gli algoritmi che sono stati utilizzati per generare i livelli del videogioco. Nella Sezione 4.1 abbiamo analizzato tutti gli algoritmi che generano i livelli di ogni universo. Nella Sezione 4.2 abbiamo fatto un confronto tra la teoria generale della generazione procedurale di contenuti e quanto invece è stato messo in atto durante la realizzazione del videogioco. Nella sottosezione 4.2.1 abbiamo analizzato la classificazione degli algoritmi. Nella sottosezione 4.2.2 abbiamo trattato la rappresentazione dei contenuti. Nella sottosezione 4.2.3 abbiamo confrontato le funzioni di fitness.

5 Conclusioni e sviluppi futuri

In questo lavoro abbiamo affrontato il problema di generare dei livelli per un videogioco come *Journey at the Edge of the Universe*, un puzzle game basato sulla fisica. Abbiamo implementato e analizzato diversi algoritmi che risolvono questo problema.

5.1 Conclusioni

La generazione procedurale di contenuti permette di risparmiare molto tempo durante la progettazione di un videogioco e consente di concentrarsi maggiormente su altri aspetti, sapendo che l'algoritmo che è stato scritto genererà sicuramente dei contenuti ritenuti accettabili.
Gli algoritmi realizzati hanno dato tutti un risultato positivo. I livelli generati sono risultati giocabili e non si sono verificate situazioni indesiderate in cui il giocatore non riusciva a comandare il razzo. L'algoritmo del primo e del secondo universo è il più semplice tra quelli realizzati e riesce a generare dei livelli topologicamente diversi ad ogni esecuzione. L'algoritmo del terzo universo è una estensione del precedente e posiziona correttamente degli ostacoli che il giocatore deve evitare. L'algoritmo del quarto universo è quello che ha il maggior potenziale creativo perchè gli elementi di cui dispone per generare un livello sono molti di più degli elementi a disposizione negli universi precedenti. Infine l'algoritmo del quinto universo torna a generare un solo tipo di elemento, il pianeta con satelliti, ma la variazione più rilevante riguarda la modalità utilizzata per disporre i pianeti nell'ambiente di gioco. Quest'ultimo algoritmo è quello che richiede il maggior tempo di esecuzione perchè avviene di frequente che si cerchi di sovrapporre due pianeti e quindi è necessario riposizionare uno dei due al fine di garantire la giocabilità del livello.

5.2 Sviluppi futuri

Il principale sviluppo futuro di questo videogioco è trovare una **grafica** che gli permetta di risultare più gradevole e che consenta al giocatore di capire con maggiore facilità quali sono gli elementi che vede sullo schermo e quali sono le loro caratteristiche. Vedere un vero razzo muoversi o un vero pianeta ruotare è molto più intuitivo rispetto a vedere una sfera rossa e una sfera bianca. Dopo aver aggiunto a *Journey at the Edge of the Universe* una grafica adatta, il passo successivo potrebbe già essere quello della **pubblicazione** sui principali store di applicazioni mobile. Unity, il software utilizzato per realizzare il videogioco, consente di produrre giochi che possono essere eseguiti su moltissime piattaforme. Il target principale a cui ci rivolgeremmo sarebbero gli smartphone e i tablet con sistemi operativi Android e iOS. Il gioco ha un livello di maturità abbastanza buono e questo ci permetterebbe di farlo testare da molte persone.

Se invece si volesse nuovamente intervenire sul codice del videogioco, sarebbe possibile modificare gli algoritmi che generano i livelli. Invece di utilizzare l'attuale funzione di fitness diretta, si potrebbe realizzare una funzione di fitness basata sulla simulazione oppure di tipo interattivo. La funzione di fitness basata sulla simulazione sarebbe la più complessa da realizzare perchè richiederebbe la creazione di un agente in grado di giocare. La funzione di fitness interattiva invece potrebbe essere una soluzione più percorribile. Sarebbe infatti sufficiente organizzare delle sedute di test in cui alcune persone verrebbero intervistate per avere un'idea più precisa di cosa i giocatori stanno cercando, mentre altre persone potrebbero giocare a *Journey at the Edge of the Universe* permettendo di

raccogliere dei dati su quali sono le maggiori difficoltà e su come interagisce con i vari elementi di gioco.
Si potrebbero aumentare i parametri che compongono il genotipo, ad esempio aggiungendo il numero di volte che il razzo è stato distrutto oppure il tempo impiegato dal giocatore per terminare il livello. Utilizzando questi parametri sarà possibile creare il livello successivo basandosi anche su dei valori che provengono dal giocatore stesso e che quindi spingeranno l'algoritmo verso la creazione di un livello più semplice oppure più complesso.

Inoltre sarebbe possibile ampliare il numero degli universi di gioco, unendo gli algoritmi già esistenti. Sarebbe possibile creare un nuovo universo unendo gli elementi utilizzati dall'algoritmo del quarto universo con il posizionamento realizzato dall'algoritmo del quinto universo. Oppure si potrebbe creare un universo con le caratteristiche del quinto universo, ma con in aggiunta gli asteroidi del terzo universo che selezionano un percorso da seguire.

Bibliografia

[1] Gillian Smith,
"Procedural Everything": Playing Procedural Content Generation in Spore
Well-Played Symposium, co-located with the Digital Games Research Association Conference (DiGRA 2014), Snowbird, UT, August 3-6, 2014.

[2] Mohammad Shaker, Noor Shaker, Julian Togelius,
Evolving Playable Content for Cut the Rope through a Simulation-Based Approach
Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-13, Boston, Massachusetts, USA, October 14-18, 2013

[3] Mohammad Shaker, Mhd Sarhan, Ola Al Naameh, Noor Shaker, Julian Togelius,
Automatic Generation and Analysis of Phisics-Based Puzzle Games
2013 IEEE Conference on Computational Inteligence in Games (CIG), Niagara Falls, ON, Canada, August 11-13, 2013

[4] Julian Togelius, Georgios N. Yannakis, Kenneth Stanley, Cameron Browne,
Search-Based Procedural Content Generation: A Taxonomy and Survey
IEEE Trans. Comput. Intellig. and AI in Games 3(3): 172-186 (2011)

[5] Georgios N. Yannakis, Julian Togelius,
Experience Driven Procedural Content Generation
T. Affective Computing 2(3): 147-161 (2011)

[6] Ahmed Khalifa, Magda Fayek,
Automatic Puzzle Level Generation: A General Approach using Description Language
Computational Creativity & Games Workshop, Park City, UT, June 28th, 2015

[7] Noor Shaker, Julian Togelius, Georgios N. Yannakis, Ben Weber, Tomoyuki Shimizu, Tomonori Hashiyama, Nathan Sorenson, Philippe Pasquier, Peter Mawhorter, Glen Takahashi, Gillian Smith, Robin Baumgarten,
The 2010 Mario AI Championship: Level Generation Track
IEEE Trans. Comput. Intellig. and AI in Games 3(4): 332-347 (2011)

[8] Adam M. Smith, Erik Andersen, Michael Mateas, Zoran Popovic,
A Case Study of Expressively Constrainable Level Design Automation Tools for a Puzzle Game
International Conference on the Foundations of Digital Games, FDG '12, Raleigh, NC, USA, May 29 - June 01, 2012: 156-163

[9] Leslie Pack Kaelbling, Michael L. Littman, Andrew W. Moore,
Reinforcement Learning: A Survey
J. Artif. Intell. Res. (JAIR) 4: 237-285 (1996)

[10] Steve Dahlskog, Julian Togelius,
Patterns and Procedural Content Generation
Proceedings of the FDG Workshop on Design Patterns in Games (DPG) 2012

[11] Yoshio Murase, Hitoshi Matsuraba, Yuruzu Hiraga,
Automatic Making of Sokoban Problems
PRICAI'96: Topics in Artificial Intelligence, 4th Pacific Rim International Conference on Artificial Intelligence, Cairns, Australia, August 26-30, 1996: 592-600

[12] Tomas Rychnovsky,
Procedural Generation of Puzzle Game Levels
http://www.gamedev.net/page/resources/_/technical/game-programming/procedural-generation-of-puzzle-game-levels-r3862

[13] Mohammad Shaker, Noor Shaker, Julian Togelius, Mohamed Abou-Zleikha,
A Progressive Approach to Content Generation
EvoApplications 2015, Copenhagen, Denmark: 381-393

[14] Fionda Gravitazionale
https://en.wikipedia.org/wiki/Gravity_assist

[15] Wormhole
https://en.wikipedia.org/wiki/Wormhole

[16] Buco Nero
https://en.wikipedia.org/wiki/Black_Hole

[17] Unity
https://www.unity3d.com

[18] Monodevelop
https://www.monodevelop.com

[19] GitHub
https://github.com

[20] Android SDK
https://developer.android.com/sdk/

[21] JDK
https://www.oracle.com/technetwork/java/javase/downloads/index.html