UNIVERSITÀ DEGLI STUDI DI ROMA
TOR VERGATA

FACOLTÀ DI SCIENZE MM. FF. NN.

CORSO DI LAUREA TRIENNALE IN INFORMATICA

A.A. 2020/2021

Tesi di Laurea

PAC-MAN AP

Reinforcement learning applicato al videogioco

RELATORE

Prof. Roberto Basili

CANDIDATO

Emanuele Izzo
0253052

CORRELATORI

Prof. Danilo Croce

A mia sorella, la mia colonna portante,

grazie, per avermi sempre sostenuto.

A mia nonna, che mi guarda da lassù,

sperando tu sia fiera di quello che ho realizzato.

Indice

Ringraziamenti

Non posso che esprimere la ma graditudine, in primis, per il prof. **Roberto Basili** e il prof. **Danilo Croce**, che mi hanno supportato (e sopportato), e che mi hanno permesso di presentare questo progetto molto particolare, ma che mi permette di raccontare le mie passioni.

Un grazie ENORME va soprattutto a **Luigi**, **Ilaria**, **Federica**, **Alessia**, **Beatrice** ed altri, che hanno sopportato tutti i miei scleri universitari e non, rimanendomi.

Un grazie speciale va anche ai miei compagni di corso, tra cui **Riccardo**, **Andrea D.**, **Gianmarco C.**, **Matteo D.S.**, e tutta quella combriccola di pazzi che rientra nella categoria "amici dell'università", che mi hanno supportato (e sopportato), alleggerendo questa pazza avventura che è la triennale.

Un ringraziamento speciale va anche a **mia madre**, **mia sorella Tamara**, **Dino** e **Dina**, che mi sono sempre state accanto, sia nei momenti di difficoltà, sia quando ho fatto cavolate apocalittiche, e sia quando ho avuto delle "crisi" a causa degli esami.

Chiudo con la mia più grande riconoscenza a **Roberta** ed **Elisa**, che mi hanno permesso di far parte di un progetto bellissimo, e con cui spero di poter realizzare ulteriori progetti.

Introduzione

Gli algoritmi di reinforcement learning sono attraenti per imparare a controllare un agente in ambienti diversi. Esistono alcune applicazioni dell'apprendimento per rinforzo di grande successo, tra cui *AlphaGo*, sviluppato da Google DeepMind per il gioco del go. Tale software utilizza una combinazione di machine learning e tecniche di ricerche su alberi, sfruttando l'apprendimento supervisionato (basato sul gioco umano) e l'apprendimento per rinforzo. Un ulteriore esempio è l'evoluzione di AlphaGo, ossia *AlphaZero* sviluppato sempre da Google DeepMind ma in grado di effettuare a partite di shogi, go e scacchi. COme il suo predecessore, utilizza la ricerca ad albero guidata da una rete neurale convoluzionale profonda addestrata per rinforzo. Non solo state sviluppate solo AI in grado di effettuare partite a giochi da tavolo: alcuni ricercatori dell'università di Toronto hanno sviluppato un AI che, tramite al Deep Reinforcement Learning, riesce ad effettuare partite ad alcuni giochi dell'Atari, tra cui Pong, *Breakout*, *Space Invaders*, *Seaquest* e *Beam Rider*. La loro AI si basa sull'elaborazione della schermata di gioco da parte di una rete neurale convoluzionale addestrata tramite reinforcement learning. Seppur esistano ulteriori esempi rispetto a quelli citati, una delle problematiche ancora non risolte è quella di trattare efficentemente enormi spazi degli stati in modo da ottenere ottimi risultati nel minor tempo di training possibile. Uno dei video giochi più famosi al mondo che presenta uno spazio degli stati incredibilmente grande è *Pac-man*, ideato da Tōru Iwatani nel 1980. Sono varie le soluzioni che sono state proposte, aventi come fattore comune però l'utilizzo di una convolutional neural network addestrata tramite reinforcement learning, ma poche sono le soluzioni proposte che hanno mostrato risultati positivi.
Questa tesi vuole proporre una possibile struttura di un agente che, tramite una DQN addestrata tramite reinforcement learning, apprendere come effettaure una partita a Pac-Man. Il primo capitolo discute di tutte le tecnologie usate, in particolare la struttura delle rete neurali e il reinforcement learing, e la storia del gioco trattato. Nel secondo capitolo viengono invece descritte nel dettaglio le meccaniche del gioco, gene-

rali e speifiche dei fantasmi, descrivendo nel dettaglio l'alrotimo di scelta della mossa successiva dei fantasmi. Nel terzo capitolo viene presentata un'implementazione di base del gioco, con la discussione dei difetti di tale versione. Nel capitolo quattro viene presentata un'implementazione rivisitata del gioco realizzata da me, puntando il fcus sulle migliorie e le modifiche apportate. Il capitolo cinque tratta la struttura dell'agente, definendo funzione di decisione, parametri del modello, e misure prestazionali, Il capitolo sei presenta le valutazioni delle prestazioni riscontrate durante un periodo di training dell'agente, con relative considerazione dei risultati ottenuti. Sono presenti inoltre nelle appendici i codici sorgente delle varie implementazioni (base, revisione e agente).

Capitolo 1

Rassegna della tecnologia usata e storia di Pac-Man

1.1 La storia del gioco

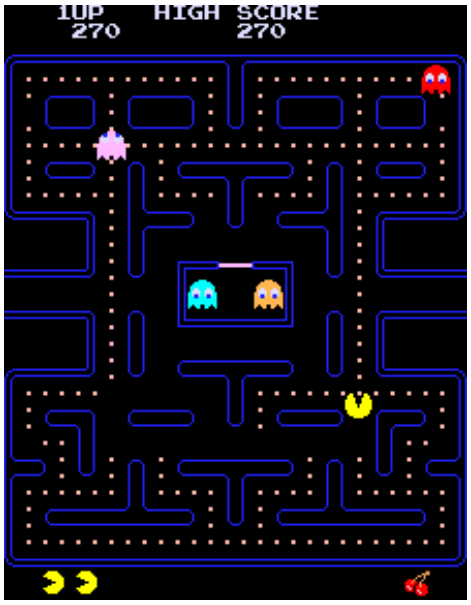

Figura 1.1: La schermata di gioco di Pac-Man.

Figura 1.2: Uno dei primi cabinati di Pac-Man.

Pac-Man è un videogioco ideato da Tōru Iwatani e prodotto dalla Namco nel 1980 nel formato arcade da sala. In occidente fu pubblicato in licenza dalla Midway Games. Acquisì subito grande popolarità e, negli anni successivi, sotto l'etichetta

Namco sono state pubblicate varie versioni per la quasi totalità delle console e dei computer, conservando fino a oggi la sua fama di classico dei videogiochi. L'origine di Pac-Man è piuttosto singolare. Sembra infatti che l'idea soggiunse a Toru Iwatani durante una cena con degli amici guardando una pizza a cui era stata tolta una fetta. Dopo quattordici mesi da quella cena, precisamente il 22 maggio del 1980, grazie a un team di sviluppo di otto tecnici, divisi equamente fra software e hardware e capeggiati da Shigeo Funaki, comprendente anche il musicista Toshio Kai, vide la luce il primo Pac-Man. Il gioco fu commercializzato in Giappone a partire dal 10 maggio con il nome di Puckman, termine che deriva dalla parola giapponese pakupaku, ovvero "chiudere e aprire la bocca". Il nome fu poi cambiato in Pac-Man per la sua commercializzazione negli Stati Uniti, iniziata nell'agosto dello stesso anno, a causa di una spiacevole assonanza con una parolaccia inglese: si temeva che "Puckman" potesse essere storpiato in un osceno "Fuckman". Nel novembre dello stesso anno Pac-Man viene presentato all'Amusement and Music Operators Association (AMOA) di Chicago dove venne definito "troppo carino per avere successo". Le previsoni dell'AMOA furono presto smentite, perché invece il successo del "mangia-palline" fu strepitoso: la Namco piazzò, in soli sette anni (dal 1980 al 1987), più di 300000 macchine e vendette milioni di gadget e pupazzi vari. Dall'uscita della prima versione del gioco, vennero rilasciati nuovi giochi, con meccaniche molto simili al Pac-Man originale, tra cui:

- *Ms. Pac-Man*, rilasciato nel 1982 da Bally Midway, utilizza le meccaniche del gioco originale, avendo però una nuova mappa come area di gioco e modificando il behaviur di due dei quattro fantasmi;
- *Super Pac-Man*, sviluppato dalla Namco e rilasciato nel 1982, dove Pac-Man, invece di mangiare tutti i dot presenti sulla mappa, deve riuscire a mangiare tutta la frutta presente, sbloccando inoltre alcune porte con delle chiavi presenti sulla mappa;
- *Jr. Pac-Man*, rilasciato nel 18983 da Bally Midway, il quale utilizza una mappa dio gioco due volte più grande di quella originale;
- *Pac & Pal*, rilasciato esclusivamente in Giappone dalla Nomco, riprende le meccaniche di Super Pac-Man, sostituendo le chiave con delle carte, e introducento un nuovo personaggio, Miru, un fantasma che aiuta il player ne completamento del livello.

- *Pac Mania*, rilasciato dalla Namco nel 1987, utilizza tutte le meccaniche del gioco originale, usando però una grafica 3D isometrica;
- *Pac-Man World*, rilasciato nel 1999 dalla Namco Hometek, è il primo platform 3D della serie, realizzato per festeggiare il 20-esimo anniversario dall'uscita di Pac-Man.

1.2 Le reti neurali

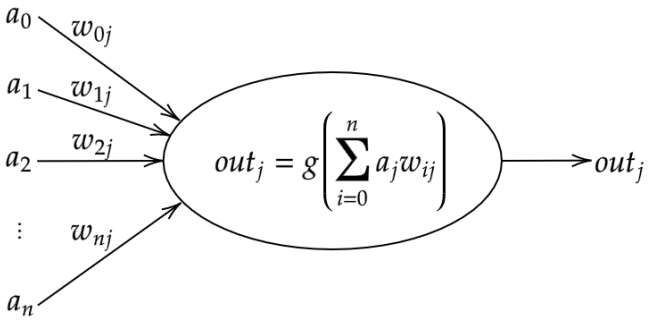

Figura 1.3: Un modello matematico semplice di un neurone.

L'attività mentale consiste principalmente nell'attività elettrochimica nelle reti di cellule cerebrali chiamate neuroni. Le **reti neurali** (**neural networks**) sono una collezione di unità connesse tra loro; le proprietà delle reti sono determinate dalla sua topologia e della proprietà dei suoi "neuroni".

1.2.1 Struttura delle reti neurali

Le reti neurali sono composte da nodi o **unità** connesse da **link** diretti. Un link da un'unità i all'unità j serve a propagare l'**attivatore**. Ogni link possiede inoltre un **peso** numerico w_{ij} associato ad esso, che determina la forza e il segno della connessione. Come un modello di una regressione lineare, ogni unità possiede un input dummy $a_0 = 1$ con un peso associato w_{0j}. Ogni unità j esegue prima una somma pesata degli input, per poi applicare la **funzione di attivazione** g su questa somma per ottenere l'output:

$$out_j = g(in_j) = g(\sum_{i=0}^{n} a_i w_{ij})$$

La funzione di attivazione g è tipicamente un hard threshold, in tal caso l'unità è detta **perceptron**, o una funzione logistica, in tal caso viene spesso usato il termine **sigmoid perceptron**. Entrambe le funzioni di attivazione non lienari assicurano l'importante proprietà che l'intera rete di unità può rappresentare una funzione non lineare. Una **rete feed-forward** (**feed-forward network**) ha connessioni tutte dirette in una sola direzione, formando un grafo diretto aciclico; rappresenta pertanto una funzione basata sul solo input, e pertanto non possiede alcuno stato interno se non i pesi stessi. Una **rete ricorrente** (**recurrent network**), d'altra parte, usa l'output prodotto come input all'interno della rete; questo significa che il livello di attivazione della rete forma un sistema dinamico che può raggiungere uno stato stabile o mostrare oscillazioni o anche presentare un comportamento caotico. Inoltre, la risposta della rete dato un certo input dipende dal suo stato iniziale, che può dipendere del precedente input, pertanto le reti ricorrenti possono supportare memoria a breve termine.Le reti feed-forward sono di solito organizzate su **strati** (**layers**), e una rete a più strati possiede uno o più strati di **unità nascoste** (**hidden units**) che non sono connesse all'output della rete.

1.2.2 Reti neurali feed-forward a singolo strato (perceptrons)

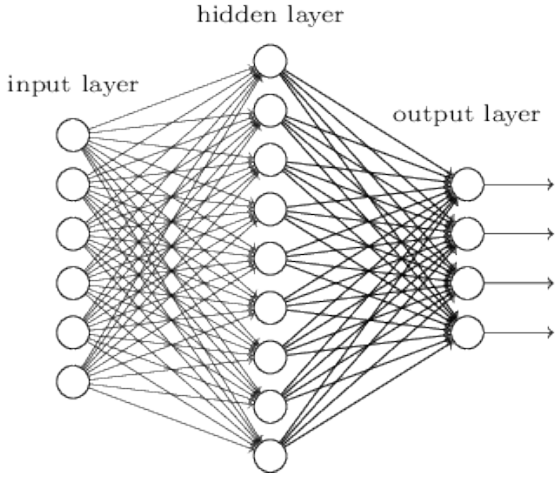

Figura 1.4: Una rete neurale feed-forward a singolo strato.

Una rete con tutti gli input connessi direttamente agli output è detta **rete neurale a singolo strato** (**single-layer neural network**) o **perceptron** (*percettrone*. Il percettrone fu proposto da Frank Rosenblatt nel 1958 come un'entità con uno strato di ingresso, uno di uscita ed una regola di apprendimento basata sulla minimizzazione

dell'errore. La prima cosa da notare è che un perceptron network con m output è in realtà composto da m reti separate, dato che ogni peso lavora su uno solo degli output. pertanto, ci saranno m processi di training separati. inoltre, in base alla funzione di attivazione usata, il processo di training sarà la **perceptron learning rule** o regola di discesa del gradiente delle **regressione logistica**.

1.2.3 Reti neurali feed-forward a più strati e apprendimento

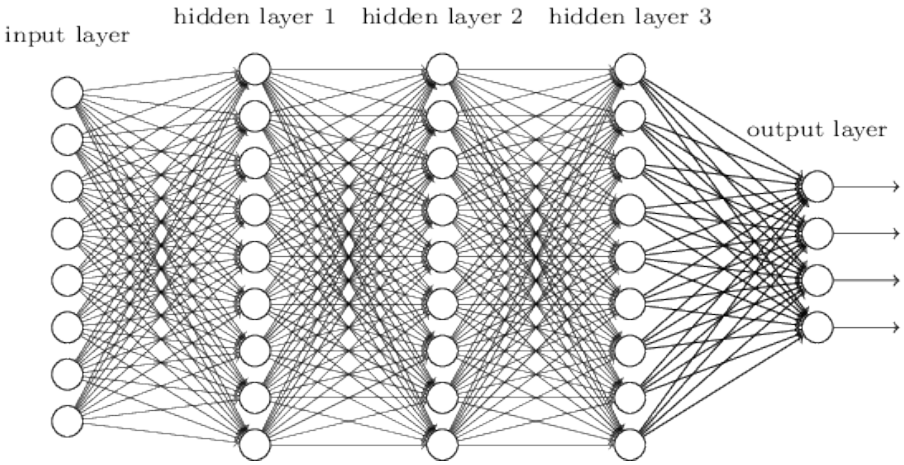

Figura 1.5: Una rete neurale feed-forward a più strati.

Allenare questa rete risulta semplice se pensiamo la rete nel modo giusto: come una funzione $h_{\mathrm{w}}(\mathrm{x})$ parametrizzata dai pesi w. Dato che la funzione rappresentata dalla reta può essere molto non lineare, composta quindi da soft threshold function nidificate, possiamo vedere la rete neurale come un mezzo per effettuare una **regressione non lineare**. In caso di interazioni tra i problemi di apprendimento nel caso che la rete possiede output multipli, la rete andrebbe vista come l'implementazione di una funzione vettore $\mathrm{h_w(x)}$ piuttosto che una funzione scalare $h_{\mathrm{w}}(\mathrm{x})$: in questo caso l'output sarà un vettore y. Seppur un perceptron network si decompone in m problemi di apprendimento separati per un problema ad m output, questa decomposizione fallisce nelle reti a più strati. Questa dipendenza è molto semplice nel caso di una qualsiasi loss function che sia *additiva* attraverso i componenti del vettore di errore $\mathrm{y} - \mathrm{h_w(x)}$. Per la L_2 loss, abbiamo, per qualsiasi peso w,

$$\frac{\delta}{\delta w} Loss(\mathrm{w}) = \frac{\delta}{\delta w}|\mathrm{y} - \mathrm{h_w(x)}|^2 = \frac{\delta}{\delta w}\sum_k (y_k - a_k)^2 = \sum_k \frac{\delta}{\delta w}(y_k - a_k)^2$$

Dove l'index k scorre su tutti i nodi dello strato di output. Ogni termine nella sommatoria finale è solo il gradiente della loss del k-esimo output, calcolato come se gli altri output non esistessero. Pertanto, possiamo decomporre un problema di apprendimento con m output in m problemi di apprendimento. L'error rate degli strati nascosti sembrano misteriosi dato che i dati di training non dicono quale valore i nodi nascosti devono avere. Possiamo **propagare all'indietro** (**back-propagate**) l'errore dallo strato di output agli strati nascosti. Sia Err_k il k-esimo elemento del vettore di errore $\mathrm{y} - \mathrm{h_w(x)}$, sia $\Delta_k = Err_k \cdot g'(in_j)$, l'aggiornamento dei pesi diventa

$$w_{jk} \leftarrow w_{jk} + \alpha \cdot a_j \cdot \Delta_k$$

Per aggiornare la connessione tra le unità d'input e le unità nascoste, dobbiamo definire una quantità analoga al termine d'errore per i nodi di output. L'idea è che il nodo nascosto j è "responsabile" di una frazione dell'errore Δ_k in ognuno dei nodi di output ai quali si connette. Pertanto, il valore Δ_k è diviso in base alla forza della connessione tra il nodo nascosto e il nodo di output ed è propagato al'indietro per fornire i valori Δ_j per gli strati nascosti. La regola di propagazione per i valori Δ_j è la seguente:

$$\Delta_j = g'(in_j) \sum_k w_{jk} \Delta_k$$

Ora la regola di aggiornamento dei pesi per i vari pesi tra gli strati di input e quelli nascosti è essenzialmente identica alla regola di aggiornamento per lo strato di output:

$$w_{ij} \leftarrow w_{ij} + \alpha \cdot a_j \cdot \Delta_j$$

Il processo di propagazione all'indietro può essere riassunto come segue:

- Calcola i valori di Δ per le unità di output, usando l'errore osservato;
- Partendo dallo strato di output, ripete il seguente procedimento per ogni strato della rete, finché non viene raggiunto il primo strato nascosto:
  - Propaga all'indietro il valore di Δ al precedente strato;
  - Aggiorna i pesi tra i due strati.

Poniamo di cacolare solo il gradiente per $Loss_k = (y_k - a_k)^2$ al k-esimo output. Il gradiente di questa loss riguardo ai pesi che connettono lo strato nascosto con lo strato

di output saranno zero tranne il peso w_{jk} che connette con la k-esima unità di output. Per questi pesi, abbiamo

$$\begin{aligned}\frac{\delta Loss_k}{\delta w_{jk}} &= -2(y_k - a_k)\frac{\delta a_k}{\delta w_{jk}} = -2(y_k - a_k)\frac{\delta g(in_k)}{\delta w_{jk}} = \\ &= -2(y_k - a_k)g'(in_k)\frac{\delta in_k}{\delta w_{jk}} = -2(y_k - a_k)g'(in_k)\frac{\delta}{\delta w_{jk}}(\sum_j w_{jk}a_j) = \\ &= -2(y_k - a_k)g'(in_k)a_j = -a_j\Delta_k\end{aligned}$$

Per ottenere il gradiente riguardo ai pesi w_ij che connetto lo strato di input allo strato nascosto, dobbiamo espandere gli attivatori a_j:

$$\begin{aligned}\frac{\delta Loss_k}{\delta w_{ij}} &= -2(y_k - a_k)\frac{\delta a_k}{\delta w_{ij}} = -2(y_k - a_k)\frac{\delta g(in_k)}{\delta w_{ij}} = \\ &= -2(y_k - a_k)g'(in_k)\frac{\delta in_k}{\delta w_{ij}} = -2\Delta_k\frac{\delta}{\delta w_{jk}}(\sum_j w_{jk}a_j) = \\ &= -2\Delta_k w_{jk}\frac{\delta a_j}{\delta w_{jk}} = -2\Delta_k w_{jk}\frac{\delta g(in_j)}{\delta w_{jk}} = \\ &= -2\Delta_k w_{jk}g'(in_j)\frac{\delta in_j}{\delta w_{jk}} = \\ &= -2\Delta_k w_{jk}g'(in_j)\frac{\delta}{\delta w_{jk}}(\sum_j w_{ij}a_i) = \\ &= -2\Delta_k w_{jk}g'(in_j)a_i = -a_i\Delta_j\end{aligned}$$

1.2.4 Apprendere la struttura delle reti neurali

Le reti neurali sono soggette all'**overfitting** quando sono presenti troppi parametri nel modello. Se ci atteniamo alle reti neurali completamente connesse, l'unica scelta da fare riguarda il numero di strati nascosti e la loro grandezza. L'approccio usuale è quello di fare varie prove e tenere da parte i risultati migliori. Le tecniche di **cross validation**, ossia tecniche di convalida di modelli predittivi per stimare quanto accuratamente un modello predittivo funzionerà, sono necessarie per evitare il **peeking** al set di test, ossia dell'utilizzo del set di test sia per scegliere un'ipotesi che per valutarla. Se vogliamo considerare reti che non sono completamente connesse, allora dobbiamo trovare metodi efficaci all'intero dell'enorme spazio delle possibili topologie di connessione, L'algoritmo di **tiling**, per esempio, ricorda il decision-list learning, e sfrutta l'idea di partire con una sola unità che fà del suo meglio per produrre il

risultato corretto sul maggior numero di esempi di training possibile, per poi aggiungere sequenzialmente varie unità per occuparsi dei vari esempi che la prima unità ha sbagliato.

1.3 Apprendimento per rinforzo

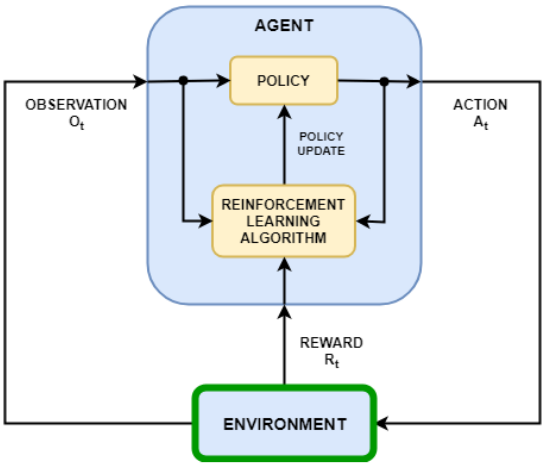

Figura 1.6: Schema del reinforcement learning

Con **reward** (**ricompensa**) o **rifonzo** si indica un feedback fornito all'agente per indicargli se ha raggiunto un esito positivo o negativo. il compito dell'**apprendimento per rinforzo** (**reinforcement learning**) è di usare le reward osservate per apprendere una policy ottima (o quasi ottima) per l'ambiente. L'apprendimento per rinforzo può essere considerato per incoraggiare tutte le IA: un agente è piazzato in un ambiente e deve imparare a comportarsi con successo al suo interno.

1.3.1 Apprendimento per rinforzo passivo

Iniziamo dal caso di un passive learning agent usando una rappresentazione basata sugli stati in un ambiente completamente osservabile. Nell'apprendimento passivo, la policy dell'agente π è fissa: in uno stato s, viene sempre eseguita l'azione $\pi(s)$. Il suo obiettivo è semplicemente imparare quanto è buona la policy, ossia imparare la funzione di utility $U^{\pi}(s)$. Chiaramente, l'attività di apprendimento passivo è simile all'attività di **valutazione delle policy**. La differenza principale è che il passive learning agent non conosce il **modello di transizione** $P(s|s,a)$, che specifica la probabilità di raggiungere uno stato s' dallo stato s dopo aver effettuato un'azione a;

non conosce neanche la **funzione di reward** $R(s)$, che specifica la reward per ogni stato. L'agente esegue una serie di **prove** nell'ambiente usando la policy π. Lo scopo è quello di usare le informazioni sulle reward per imparare l'utility prevista $U^{pi}(s)$ associata ad ogni stato non deterministico s. L'utility è definita come la somma prevista delle reward (scontate) ottenute se viene seguita la policy π:

$$U^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t R(S_t)]$$

Dove $R(s)$ è la reward per uno stato, S_t (una variabile casuale) è lo stato raggiunto al tempo t quando viene eseguita la policy π, e $S_0 = s$. Includeremo un **fattore di sconto** γ in tutte le nostre equazioni.

Stima diretta dell'utility

Un metodo semplice per la **stima diretta dell'utility** è stato inventato alla fine degli anni '50 nell'area della **teoria del controllo adattivo** da Widrow e Hoff (1960). L'idea è che l'utility di uno stato è il reward totale previsto da quello stato in poi (chiamata la **reward-to-go** prevista), e ogni prova fornisce un *campione* di questa quantità per ogni stato visitato. Alla fine di ogni sequenza, l'algoritmo calcola la reward-to-go osservata per ogni stato e aggiorna l'utility prevista per ogni stato in accordo, semplicemente mantenendo una media corrente per ogni stato in una tabella. Nel limite di infinite prove diverse, la media dei campioni convergerà al valore atteso. È chiaro che la stima diretta dell'utility è solo un istanza dell'apprendimento supervisionato dove ogni esempio a lo stato come input e la reward-to-go osservata come output appunto questo significa che abbiamo ridotto l'apprendimento per rinforzo ha un problema standard di apprendimento induttivo. La stima diretta dell' utility succede nel ridurre il problema di apprendimento per rinforzo in un problema di apprendimento induttivo, per quello che noi sappiamo. Sfortuna nettamente, non considera una fonte di informazioni molto importante, ossia il fatto che le utility degli stati non sono indipendenti. L' utility di ogni stato è uguale alla sua reward più l'utility prevista per il suo stato successivo, come indicato nell'equazione di Bellman:

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^{\pi}(s')$$

Ignorando la connessione tra gli stati, la stima diretta dell'utility perde occasione per l'apprendimento.

Adaptive dynamic programming

Un **adaptive dynamic programming** (o **ADP**) agent prende il vantaggio dei vincoli insieme alle utility degli stati prendendo il modello di transizione che li connette e risolvendo il corrispettivo processo decisionale di Markov usando un metodo di programmazione dinamica. Per un passive learning agent, questo significa inserire il modello di transizione appreso e le reward osservate all'interno dell'equazione di Bellman per calcolare le utility degli stati. Queste equazioni sono lineari (non involgono massimizzazione) quindi possono essere risolte utilizzando un qualsiasi package di algebra lineare. Alternativamente, possiamo adottare l'approccio di **iterazione della policy modificata** (**modified policy iteration**), usando un processo di iterazione del valore semplificato per aggiornare le stime dell'utility dopo ogni cambio sul modello ha appreso. Poiché il modello di solito cambia soltanto leggermente con ogni osservazione, il processo di iterazione del valore può usare le precedenti stime delle utility come valori iniziali e dovrebbe convergere abbastanza velocemente. Il processo di apprendimento del modello stesso è semplice, poiché l'ambiente è completamente osservabile. Questo significa che abbiamo un task di apprendimento supervisionato dove l'input è una copia stato-azione e l'output è lo stato risultante. Nel caso più semplice, possiamo rappresentare il modello di transizione che come una tabella di probabilità. Teniamo traccia di quanto spesso ogni risultato di un'azione occorre e stimare la probabilità di transizione $P(s'|s,a)$ dalla frequenza con cui s' è raggiunto quando viene eseguito a in s. In termini di quanto velocemente il valore previsto migliora, l'ADP agent è limitato soltanto dalla sua abilità di imparare il modello di transizione. In questo senso, fornisce uno standard contro il quale misurare altri algoritmi di apprendimento rinforzato. È tuttavia intrattabile per enormi spazi di stati. L'algoritmo utilizza la stima di massima verosomiglianza per apprendere il modello di transizione; inoltre, scegliendo una policy basata solamente sul modello previsto si comporta *come se* il modello fosse corretto. Questa non è necessariamente una buona idea. L'**apprendimento per rinforzo bayesiano** (**beyesian reinforced learning**) assume una probabilità a priori $P(h)$per ogni ipotesi h riguardo a quale sia il vero modello; la proprietà a posteriori $P(h|\mathrm{e})$ è ottenuta nel solito moto tramite la regola di Bayes data dalle osservazioni finora. Poi, se l'agente ha deciso di non apprendere più, la policy ottima è quella che ritorna il valore di utility previsto più alto. Sia ${u_h}^{\pi}$ l'utility prevista, media su tutti i possibili stati di avvio, ottenuta eseguendo la policy π nel modello h. Allora avremo

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \sum_h P(h|\mathrm{e}){u_h}^\pi$$

In alcuni casi speciali, questa policy può essere anche calcolata. Però, se l'agente continuerà ad apprendere nel futuro, allora trovare una polizza ottima diventa considerabilmente più difficile, poiché la gente deve considerare gli effetti delle future osservazioni sulle sue credenze riguardo il modello di transizione. L'approccio derivato dalla **teoria del controllo robusto** consente un *set* di possibili modelli $\mathcal{H}$ e definisce come policy ottimale robusta quella che restituisce il miglior risultato nel caso peggiore su $\mathcal{H}$:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \min_h {u_h}^\pi$$

Risolvere il sottostante MDP non è l'unico modo per portare le equazioni di Bellman per influenzare l'algoritmo di apprendimento. Un altro modo è usare le transazioni osservate per aggiustare le utility degli stati osservati in modo che siano in accordo con le condizioni dell'equazione. Quando una transazione avviene da uno stato s ad uno stato s', applichiamo il seguente aggiornamento di $U^\pi(s)$:

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha(R(s) + \gamma\, U^\pi(s') - U^\pi(s))$$

Qui, α è il parametro del learning rate. Dato che questa regola di aggiornamento usa le differenze delle utility tra stadi successivi, è spesso chiamata l'equazione di **differenza dei temporale** (**temporal-difference** equation), o TD. Tutti i metodi di differenza temporale lavorano aggiustando l'utility prevista tramite l'equilibrio ideale che tiene localmente quando l'utilità prevista è corretta. Notare che gli aggiornamenti comprendono solo i successori s' osservati, mentre la condizione di equilibrio attuale coinvolge tutti i possibili stati successivi. Poiché transizione rare avvengono solo raramente, il valore medio di $U^\pi(s)$ convergerà al valore corretto. Inoltre, se cambiamo α da un valore fisso ha una funzione che decresce come il numero di volte che uno stato viene visitato cresce, allora $U^\pi(s)$ stesso convergerà al valore corretto. L'agente TD non impara velocemente come un agente ADP e mostra maggiore variabilità, ma è molto più semplice e richiede meno calcoli per osservazione. Notare che *TD non richiede un modello di transizione per eseguire gli aggiornamenti.* L' approccio ADP e l'approccio TD sono in realtà connessi. Entrambi provano a fare degli aggiustamenti locali sulle utility previste in modo da rendere ogni stato "in accordo" con i suoi successori. Una differenza è che TD aggiusta uno stato per essere d'accordo con il suo successore *osservato*, mentre ADP aggiusta lo stato per essere in accordo con *tutti*

i successori che potrebbero presentarsi, pesati dalle loro probabilità. Una differenza più importante è che mentre TD effettua un singolo aggiustamento per ogni transizione osservata, ADP effettua vari aggiustamenti quanti ne servono per restaurare le consistenze tra le utility previste U e il modello dell'ambiente P. Ogni aggiustamento fatto ADP può essere visto, dal punto di vista di TD, come il risultato una "pseudoesperienza" creata simulando il corrente modello dell'ambiente. È possibile estendere l'approccio TD per usare un modello dell'ambiente per generare varie pseudoesperienze. In questo modo l'utility risultante prevista sarà sempre più vicina a quella di ADP, portando però ad un incremento del tempo di esecuzione. In modo analogo, possiamo generare versioni più efficienti di ADP approssimando direttamente l'algoritmo per l'interazione dei valori o per le iterazioni della policy. Un possibile approccio per generare velocemente risposte ragionevolmente buone è di limitare il numero di aggiustamenti fatti dopo ogni transizione osservata. Si può anche usare un'euristica per classificare i possibili aggiustamenti in modo che vengono effettuati solo quelli più importanti. la **prioritized sweeping** heuristic preferisce fare aggiustamenti a stati i cui successori *probabili* sono stati sottoposti ad un grande aggiustamento sulla loro utility prevista. Usando euristiche come queste, approssimare algoritmi ADP di solito possono imparare più o meno velocemente quanto un full ADP, in termini di numero di sequenze di transizioni, ma può essere più efficiente per vari ordini di magnitudo in termini di computazione.

1.3.2 Apprendimento per rinforzo attivo

Un agente attivo deve decidere quale azione effettuare. Prima di tutto, l'agente deve imparare un modello completo con tutte le probabilità risultanti per tutte le azioni piuttosto che il modello per policy fissata. Successivamente, dobbiamo tenere in conto il fatto che l'agente ha una scelta di azioni. L'utility che necessita di imparare sono quelle definite dalla policy ottimale:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a)U(s')$$

Questa equazione può essere risolta ottenendo la funzione di utility U usando gli algoritmi di iterazione dei valori o della policy. Dopo aver ottenuto una funzione di utility U ottimale per il modello appreso, l'agente può estrarre un'azione ottimale effettuando una previsione in avanti di un solo passo per massimizzare l'utility prevista.

Esplorazione

Un agente ADP che segue la raccomandazione della policy ottimale per il modello appreso ad ogni step *non* impara le vere utility o la vera policy ottimale: questo agente viene detto **agente goloso** (**greedy agent**). Ripetuti esperimenti mostrano che gli agenti golosi *molto raramente* convergono alla policy ottimale per questo ambiente e alcune volte convergono a delle policy veramente orrende. Il modello appreso non è lo stesso dell'ambiente reale, ciò che è ottimale nel modello appreso potrebbe essere subottimale nell'ambiente reale. L'agente non conosce come è fatto l'ambiente reale, pertanto non può calcolare l'azione ottimale per l'ambiente reale. Quello che l'agente goloso ha trascurato è che le azioni fanno molto di più che fornire rewards in accordo al modello appreso corrente, contribuiscono anche ad apprendere il modello reale influenzando le percezioni che sono ricevute. Un agente deve fare un compromesso tra lo **sfruttamento** (**exploitation**) per massimare la reward, come riflesso dalle stime correnti dell'utility, e l'**esplorazione** (**exploration**) per massimizzare il benessere a lungo termine. Il puro sfruttamento rischia di bloccarsi in un solco. La pura esplorazione per migliorare la conoscenza non è di aiuto se non viene mai messa in pratica. Uno schema *ragionevole* che porterà eventualmente ad un comportamento ottimale dall'agente deve essere goloso nel limite dell'esplorazione infinita, o **GLIE**. Uno schema GLIE deve provare ogni azione in ogni stato un numero di volte illimitato per evitare di avere una probabilità finita che un'azione ottimale è mancata a causa di una serie inusuale di risultati. Un agente ADP che usa tale scheda apprenderà eventualmente il modello corretto dell'ambiente. Una schema GLIE deve anche eventualmente diventare goloso, in modo che le azioni dell'agente diventano ottimali rispetto al modello appreso. Esistono vari schemi GLIE; uno dei più semplici è di avere che un agente sceglie un'azione casuale in una frazione di tempo $\frac{1}{t}$ e seguire la policy golosa altrimenti. Seppur convergerà eventualmente ad una policy ottimale, può richiedere risultare molto lento. Un approccio molto più sensibile darà alcuni pesi alle azioni che l'agente non ha provato molto spesso, mentre tenderà ad evitare azioni che sono credute essere di poca utilità. Usiamo $U^+(s)$ per indicare la stima ottimistica dell'utility dello stato s, e sia $N(s,a)$ il numero di volte l'azione a è stata provata nello stato s. Supponiamo che stiamo usando un'iterazione del valore in un agente ADP che apprende. La sua equazione di aggiornamento sarà:

$$U^+(s) \leftarrow R(s) + \gamma \max_a f(\sum_{s'} P(s'|s,a)U^+(s'), N(s,a))$$

Dove $f(u, n)$ è detta la **funzione di esplorazione**. Determina quanto la golosità (preferenza per alti valori di u) viene scambiata con la curiosità (preferenza per le azioni che non sono state provate spesso e hanno un valore n basso). La funzione $f(u, n)$ dovrebbe essere crescente in u e decrescente in n. Esistono varie possibili funzioni che soddisfano queste condizioni. Una definizione particolare e molto semplice è

$$f(u, n) = \begin{cases} R^+ & \text{se } n < N_e \\ u & \text{altrimenti} \end{cases}$$

Dove R^+ è una stima ottimistica della miglior reward possibile ottenibile in qualsiasi stato e N_e è un parametro fissato. Questo avrà l'effetto di portare l'agente a provare coppia stato-azione almeno N_e volte. Il fatto che U^+ appare nella parte destra dell'equazione piuttosto che U è molto importante. Mentre l'esplorazione procede, gli stati e le azioni vicini allo stato iniziale saranno stati provati un gran numero di volte. Se usassimo U, la stima delle utility più pessimista, allora l'agente diventerebbe incline a non esplorare più lontano. L'uso di U^+ significa che che i benefit dell'esplorazione sono propagati all'indietro dai confini delle regioni non esplorate, cosicché le azioni che portano *in direzione* di zone inesplorate hanno pesi maggiori, rispetto alle azioni che sono loro stesso non familiari.

Apprendere un'action-utility function

Un active temporal-difference learning agent non è più equipaggiato con un policy fissata, pertanto, se apprenderà una funzione di utility U necessiterà di apprendere un modello in modo di poter scegliere un'azione in base ad U effettuando una previsione in avanti di un solo passo. Il problema di acquisizione del modello per l'agente TD è identico a quello per l'agente ADP. La regola di aggiornamento del TD rimane invariata. Si può mostrare che l'algoritmo TD convergerà agli stessi valori di ADP se il numero di sequenze di training tende all'infinito. Esiste un metodo TD alternativo, chiamato **Q-learning**, il quale apprende una rappresentazione azione-utility invece di apprendere le utility. Useremo la notazione $Q(s, a)$ per indicare l'effettuazione di un'azione a nello stato s. I Q-values sono direttamente correlati ai valori di utility come segue:

$$U(s) = \max_a Q(s, a)$$

Le Q-function hanno una proprietà davvero interessante: *un agente TD che apprende una Q-function non necessita di un modello nella forma $P(s'|s, a)$, sia per apprendere*

che per scegliere un'azione. Per questa ragione, Q-learning è detto modello **model-free**. Per quello che riguarda le utility, possiamo scrivere un'equazione di vincolo che deve mantenere un equilibrio quando i Q-values sono corretti:

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a} Q(s',a')$$

Come nell'agente ADP, possiamo usare quest'equazione direttamente come un'equazione di aggiornamento per un processo di iterazione che calcola i Q-values esatti, dato un modello stimato. Questo, però, richiede che venga appreso un modello, poiché l'equazione usa $P(s'|s,a)$. Gli approcci a differenza temporale, d'altro canto, non richiedono un modello per le transizioni di stato, poiché necessitano solo dei Q-values. L'equazione di aggiornamento per il TD Q-learning è

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

La quale viene calcolata ogni volta che un'azione a viene eseguita nello stato s portando allo stato s'. Q-learning ha un parente chiamato **SARSA** (per State-Action-Reward-State-Action). La regola di aggiornamento per SARSA è

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max Q(s',a') - Q(s,a))$$

Dove a' è l'azione *effettivamente eseguita* nello stato s'. Questa regola è applicata alla fine di ogni quintupla s, a, r, s', a' (da qui prende il nome). La differenza dal Q-learning è abbastanza sottile: mentre Q-learning propaga all'indietro il *miglior* Q-value dallo stato raggiunto nella transizione osservata, SARSA aspetta finché non viene effettivamente eseguita un'azione e propaga all'indietro il Q-value per quell'azione. Poiché Q-learning usa il miglio Q-value, non presta attenzione alla policy attuale che viene seguita, pertanto è un algoritmo di apprendimento **off-policy**, mentre SARSA è un algoritmo **on-policy**. Q-learning è molto più flessibile del SARSA, nel senso che un agente Q-learning può imparare come comportarsi bene anche quando è guidato da un policy di eslorazione casuale o avversario. D'altro canto, SARSA è molto più realistico: è meglio imparare una Q-function per quello che succederà effettivamente piuttosto che quello che l'agente vorrebbe che accadesse. Sia Q-learning che SARSA apprendono una policy ottimale, ma lo fanno con una velocità molto più lenta di un agente ADP. Questo perché gli aggiornamenti locali non rafforzano la coerenza tra tutti i Q-values tramite il modello.

1.3.3 Generalizzazione nell'apprendimento per rinforzo

Abbiamo assunto che le funzioni di utility e le Q-function apprese dagli agenti sono rappresentate in forma tabellare con un output per ogni tupla in input. Tale approccio lavora ragionevolmente bene per spazi di stati molto piccoli, ma il tempo di convergenza e (per gli ADP) il tempo per l'iterazione aumenta rapidamente man mano che lo spazio degli stati aumenta. Un modo per trattare tali problemi è usare l'**approssimazione della funzione**, che significa semplicemente usare un qualsiasi metodo di rappresentazione per la Q-function piuttosto che la ricerca nella tabella. La funzione o Q-function può essere rappresentata nella forma scelta. Una **funzione di valutazione** è rappresentata come una funzione lneare pesata di un set di **features** (o **funzioni basi**) $f_1, f_2, ..., f_n$:

$$\hat{U}_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + ... + \theta_n f_n(s)$$

Un algoritmo di apprendimento per rinforzo può apprendere dei valori per i parametri $\theta = \theta_1, \theta_2, ..., \theta_n$ in modo che la funzione di valutazione $\hat{U}_\theta$ approssima la funzione di utility reale. L'approssimazione della funzione rende pratico rappresentare la funzione di utility per enormi spazi degli stati, ma non è il suo vantaggio principale. *La compressione ottenuta da un approssimatore della funzione permette all'agente che sta apprendendo di generalizzare dagli stati che ha visitato agli stati che non ha visitato.* L'aspetto più importante dell'approssimazione della funzione non è che richiede meno spazio, ma che permette una generalizzazione induttiva sugli stati di input. C'è il problema che potrebbe non esserci alcuna funzione nello spazio delle ipotesi scelto che approssima sufficientemente bene la funzione di utility reale. Come in tutti gli apprendimenti induttivi, esiste un tradeoff tra la grandezza dello spazio delle ipotesi e il tempo necessario ad apprendere la funzione. Uno spazio delle ipotesi molto grande incrementa la possibilità di trovare una buona approssimazione, ma significa anche che molto probabilmente la convergenza sarà ritardata. La stima diretta dell'utility con l'approssimazione della funzione è un'istanza dell'**apprendimento supervisionato**. Data una collezione di prove, otteniamo un set di valori d'esempio di $\hat{U}_\theta(x, y)$, e possiamo trovare il più adatto, nel senso di minimizzare lo squared error, usando la regressione lineare standard. Per l'apprendimento per rinforzo, ha più senso usare un algoritmo di apprendimento *online* che aggiorna i parametri dopo ogni prova. Come con l'apprendimento delle reti neurali, scriviamo un'error function e calcoliamo il suo gradiente rispetto ai parametri. Se $u_j(s)$ è la reward totale osservata dallo stato s in avanti nella j-esima prova, allora l'errore è definito come la metà della differenza

al quadrato tra il totale predetto e il totale attuale: $E_j(s) = \frac{(\hat{U}_\theta(s) - u_j(s))^2}{2}$. Il rapporto di cambio dell'errore rispetto ad ogni parametro θ_i è $\frac{\delta E_j}{\delta \theta_i}$, pertanto per muovere il parametro in modo da diminuire l'errore vogliamo

$$\theta_i \leftarrow \theta_i - \alpha \frac{\delta E_j}{\delta \theta_i} = \theta_i + \alpha(u_j(s) - \hat{U}_\theta(s)) \frac{\delta \hat{U}_\theta(s)}{\delta \theta_i}$$

Questa è detta la **regola di Widrof-Hoff**, o **regola delta**, per minimi quadrati online. Notare che *cambiare i parametri θ in risposta ad una transizione osservata tra due stati cambia anche i valori di $\hat{U}_\theta$ per ogni altro stato.* Ci aspettiamo che l'agente apprenda velocemente se usa un approssimatore della funzione, fornito in modo che lo spazio delle ipotesi non sia troppo grande, ma includa alcune funzioni che sono ragionevolmente una buon adattamento della funzione di utility reale. Ciò che importa per l'approssimazione di funzioni lineari è che la funzione sia lineare nei *parametri*, pertanto le feature possono essere arbitrariamente funzioni non lineari per le variabili di stato. Possiamo applicare queste idee altrettanto bene ai temporal-difference learners. Tutto ciò che dobbiamo fare è aggiustare i parametri per provare a ridurre la differenza temporale tra gli stati successivi. La nuova versione delle equazioni di TD e del Q-learning, rispettivamente per le utility e per i Q-values, sono date da

$$\theta_i \leftarrow \theta_i + \alpha[R(s) + \gamma \hat{U}_\theta(s') - \hat{U}_\theta(s)] \frac{\delta \hat{U}_\theta(s)}{\delta \theta_i}$$

$$\theta_i \leftarrow \theta_i + \alpha[R(s) + \gamma \max_{a'} \hat{Q}_\theta(s', a') - \hat{Q}_\theta(s, a)] \frac{\delta \hat{Q}_\theta(s, a)}{\delta \theta_i}$$

Per l'apprendimento TD passivo, si può mostrare che la regola di aggiornamento converge all'approssimazione più vicina possibile della funzione reale quando l'approssimatore della funzione è *lineare* nei parametri. Con l'apprendimento attivo e funzioni *non lineari* come le reti neurali, tutte le scommesse sono chiuse. Esistono algoritmi più sofisticati che evitano questi problemi, ma tutt'ora l'apprendimento per rinforzo con approssimazione generale delle funzioni rimane un'arte delicata. L'approssimazione delle funzioni può essere molto utile per imparare un modello dell'ambiente. Qualsiasi metodo di apprendimento supervisionato può essere usato, con aggiustamenti adatti per il fatto che necessitiamo di predire la descrizione completa dello stato piuttosto che una classificazione booleana o un singolo valore reale. Per un ambiente *parzialmente osservabile*, il problema di apprendimento è molto più difficile.

Capitolo 2

Le meccaniche di gioco

Lo scopo del gioco è molto semplice: il giocatore è posizionato in un labirinto riempito con cibo (raffigurato come dots o pellets) e deve mangiarli tutti in modo da passare al livello successivo. Questo compito è reso difficile dai quattro fantasmi che inseguono Pac-Man all'interno del labirinto. Se Pac-Man entra in contatto con uno dei fantasmi, il giocatore perde una vita e la posizione di Pac-Man e dei fantasmi viene resettata alle loro posizioni iniziali, anche se i dots mangiati non vengono toccati. Oltre che semplicemente evitarli, l'unica difesa di Pac-Man contro i fantasmi sono i quattro pallini "energizzanti" più grandi situati agli angoli del labirinto. Mangiarne uno fa sì che i fantasmi si spaventino e si ritirino per un breve periodo, e nei primi livelli del gioco Pac-Man può persino mangiare i fantasmi per ottenere punti bonus durante questo periodo. Un fantasma divorato non viene completamente eliminato, ma viene riportato alla sua posizione iniziale prima di riprendere il suo inseguimento. Oltre a mangiare punti e fantasmi, l'unica altra fonte di punti sono i due pezzi di frutta che compaiono durante ogni livello vicino al centro del labirinto. Il primo frutto appare quando Pac-Man ha mangiato 70 punti nel labirinto e il secondo quando ne sono stati mangiati 170. Ogni livello di Pac-Man utilizza lo stesso layout del labirinto, contenente 240 dots e 4 pellets. I tunnel che partono dai bordi sinistro e destro dello schermo fungono da scorciatoie per il lato opposto dello schermo e sono utilizzabili sia da Pac-Man che dai fantasmi, sebbene la velocità dei fantasmi sia notevolmente ridotta mentre sono in tunnel. Anche se il layout è sempre lo stesso, i livelli diventano sempre più difficili a causa delle modifiche alla velocità di Pac-Man, nonché delle modifiche alla velocità e al comportamento dei fantasmi. Dopo aver raggiunto il livello 21, non vengono apportate ulteriori modifiche alla meccanica del gioco e ogni livello dal 21 in poi è effettivamente identico.

2.1 Caratteristiche comuni dei fantasmi

Ciascuno dei fantasmi è programmato con una "personalità" individuale, un algoritmo diverso che utilizza per determinare il suo metodo di movimento attraverso il labirinto. Capire come si comporta ogni fantasma è estremamente importante per poterli evitare efficacemente. Tuttavia, prima di discutere i loro comportamenti individuali, esaminiamo prima la logica che condividono.

2.1.1 La zona di spawn

Quando un giocatore inizia una partita di Pac-Man, non viene immediatamente attaccato da tutti e quattro i fantasmi. Come mostrato nel diagramma della posizione iniziale del gioco, solo un fantasma inizia nel labirinto vero e proprio, mentre gli altri si trovano all'interno di una piccola area al centro del labirinto, spesso chiamata "casa fantasma". Tranne che all'inizio di un livello, i fantasmi torneranno in quest'area solo se vengono mangiati da un Pac-Man eccitato, o perché le loro posizioni vengono ripristinate quando Pac-Man muore. La casa dei fantasmi è altrimenti inaccessibile e non è un'area valida per Pac-Man o per i fantasmi in cui trasferirsi. I fantasmi si spostano sempre a sinistra non appena lasciano la casa dei fantasmi, ma possono invertire la direzione quasi immediatamente a causa di un effetto che verrà descritto più avanti. Le condizioni che determinano quando i tre fantasmi che iniziano all'interno della casa fantasma sono in grado di andarsene sono in realtà piuttosto complesse. Per questo motivo, li considererò al di fuori dello scopo di questo articolo, soprattutto perché diventano molto meno rilevanti dopo aver completato i primi livelli.

2.1.2 La casella target

Gran parte del design e della meccanica di Pac-Man ruotano attorno all'idea che il tabellone venga diviso in caselle quadrate di 8 x 8 pixel. La risoluzione dello schermo di Pac-Man è 224 x 288, quindi questo ci dà una dimensione totale del tabellone di 28 x 36 caselle, sebbene la maggior parte di questi non sia accessibile a Pac-Man o ai fantasmi. Come esempio dell'impatto delle tessere, si ritiene che un fantasma abbia catturato Pac-Man quando occupa la sua stessa casella. Inoltre, ogni pallino nel labirinto è al centro della propria casella. Va notato che poiché gli sprite di Pac-Man e dei fantasmi sono più grandi di una casella, non sono mai completamente contenuti in una singola casella. Per questo motivo, ai fini del gioco, si considera che il personaggio occupi qualsiasi casella contenga il suo punto centrale. Questa è una conoscenza importante

quando si evitano i fantasmi, poiché Pac-Man verrà catturato solo se un fantasma riesce a spostare il suo punto centrale nella stessa casella di Pac-Man. La chiave per comprendere il comportamento dei fantasmi è il concetto di una casella target. La maggior parte delle volte, ogni fantasma ha una casella specifica che sta cercando di raggiungere, e il suo comportamento ruota attorno al tentativo di arrivare a quella casella da quella attuale. Tutti i fantasmi usano metodi identici per viaggiare verso i loro target, ma le diverse personalità fantasma derivano dal modo individuale in cui ogni fantasma seleziona la propria casella target. Si noti che non ci sono restrizioni che una casella bersaglio deve essere effettivamente possibile raggiungere, possono (e spesso si trovano) posizionate su una casella inaccessibile e molti dei comportamenti fantasma comuni sono un risultato diretto di questa possibilità.

2.1.3 Modalità di movimento dei fantasmi

I fantasmi sono sempre in una delle tre modalità possibili: Chase, Scatter o Frightened. La modalità "normale" con i fantasmi che inseguono Pac-Man è Chase, e questa è quella in cui trascorrono la maggior parte del loro tempo. Mentre sono in modalità Chase, tutti i fantasmi usano la posizione di Pac-Man come fattore nella selezione della loro casella target, sebbene sia più significativo per alcuni fantasmi rispetto ad altri. In modalità Scatter, ogni fantasma ha una casella target fissa, ognuna delle quali si trova appena fuori da un angolo diverso del labirinto. Ciò fa sì che i quattro fantasmi si disperdano negli angoli ogni volta che si trovano in questa modalità. La modalità Frightened è unica perché i fantasmi non hanno una casella target specifica mentre si trovano in questa modalità. Invece, decidono in modo pseudocasuale quali svolte eseguire ad ogni incrocio. Un fantasma in modalità Frightened diventa anche blu scuro, si muove molto più lentamente e può essere mangiato da Pac-Man. Tuttavia, la durata della modalità Frightened si riduce man mano che il giocatore avanza attraverso i livelli ed è completamente eliminata dal livello 19 in poi. Le modifiche tra le modalità Chase e Scatter avvengono con un timer fisso: questo timer viene resettato all'inizio di ogni livello e ogni volta che si perde una vita. Il timer viene anche messo in pausa mentre i fantasmi sono in modalità Frightened, che si verifica ogni volta che Pac-Man mangia un pellet. Quando la modalità Frightened termina, i fantasmi tornano alla modalità precedente e il timer riprende da dove era stato interrotto. I fantasmi iniziano in modalità Scatter e sono definite quattro ondate di alternanza Scatter/Chase, dopodiché i fantasmi rimarranno in modalità Chase a

tempo indefinito (fino a quando il timer non viene resettato). Per il primo livello, le durate di queste fasi sono:

- Scatter per 7 secondi, quindi Chase per 20 secondi.
- Scatter per 7 secondi, quindi Chase per 20 secondi.
- Scatter per 5 secondi, quindi Chase per 20 secondi.
- Scatter per 5 secondi, quindi passa alla modalità Chase in modo permanente.

La durata di queste fasi cambia in qualche modo quando il giocatore raggiunge il livello 2, e ancora una volta quando raggiunge il livello 5. A partire dal livello 2, la terza modalità Chase si allunga considerevolmente, fino a 1033 secondi (17 minuti e 13 secondi), la successiva modalità Scatter dura solo 1/60 di secondo prima che i fantasmi procedano permanentemente alla loro modalità Chase. Le modifiche al livello 5 si aggiungono a questo, riducendo ulteriormente le prime due lunghezze di Scatter a 5 secondi e aggiungendo i 4 secondi guadagnati qui alla terza modalità Chase, allungandola a 1037 secondi (17 minuti e 17 secondi).

2.1.4 Regole base del movimento dei fantasmi

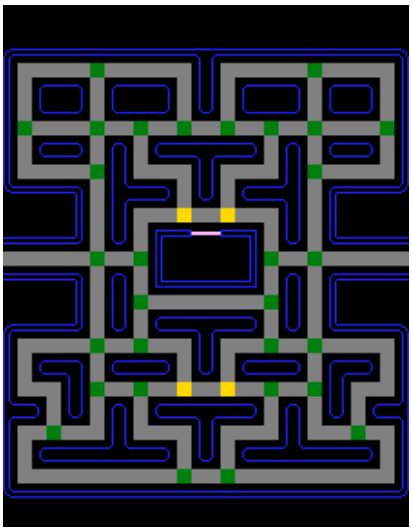

Figura 2.1: Mappa con indicate le intersezioni presenti nel labirinto

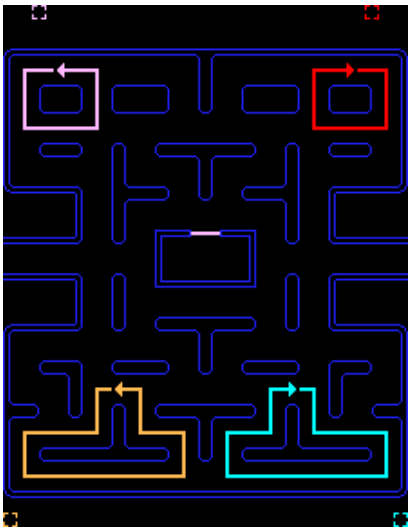

Figura 2.2: Rappresentazione della modalità Scatter

Il passo successivo è capire esattamente come i fantasmi tentano di raggiungere le loro caselle target. L'intelligenza artificiale dei fantasmi è molto semplice e miope, il che rende ancor più impressionante il loro complesso comportamento. I fantasmi pianificano solo un passo nel futuro mentre si muovono nel labirinto. Ogni volta che un fantasma entra in una nuova casella, guarda avanti alla casella successiva che raggiungerà e prende una decisione su quale direzione girerà quando ci sarà. Queste decisioni hanno una limitazione molto importante, ovvero che i fantasmi non possono mai scegliere di invertire la direzione di viaggio. Cioè, un fantasma non può entrare in una casella dal lato sinistro e poi decidere di invertire la direzione e tornare indietro a sinistra. L'implicazione di questa restrizione è che ogni volta che un fantasma entra in una casella con solo due uscite, continuerà sempre nella stessa direzione. Tuttavia, c'è un'eccezione a questa regola, ovvero che ogni volta che i fantasmi passano da Chase o Scatter a qualsiasi altra modalità, sono costretti a invertire la direzione non appena entrano nella casella successiva. Questa istruzione forzata sovrascriverà qualsiasi decisione precedentemente presa dai fantasmi sulla direzione in cui muoversi quando raggiungono quella tessera. Questo funge efficacemente da notifica al giocatore che i fantasmi hanno cambiato modalità, poiché è l'unica volta che un fantasma può invertire la direzione. Notare che quando i fantasmi lasciano la modalità Frightened non cambiano direzione, ma questo particolare passaggio è già ovvio perché i fantasmi tornano ai loro colori normali dal blu scuro di Frightened. Quindi, la modalità Scatter di 1/60 di secondo su ogni livello dopo il primo farà sì che tutti i fantasmi invertano la loro direzione di viaggio, anche se il loro obiettivo rimane effettivamente lo stesso. Questa istruzione di inversione di direzione forzata viene applicata anche a tutti i fantasmi ancora all'interno della zona di spawn, quindi un fantasma che non è ancora entrato nel labirinto quando si verifica il primo cambio di modalità uscirà dalla zona di spawn con un'istruzione "inverti la direzione non appena puoi" già in sospeso.

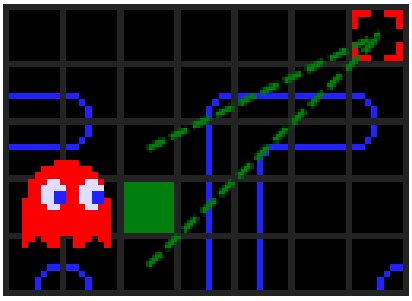

Figura 2.3: Esempio di decisione da parte di un fantasma

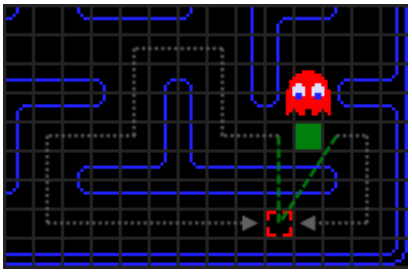

Figura 2.4: Esempio di decisione sbagliata da parte di un fantasma

Questo fa sì che si muovano a sinistra come al solito per un tempo molto breve, ma quasi immediatamente invertiranno la direzione e andranno invece a destra. L'immagine 2.1 mostra una rappresentazione semplificata del layout del labirinto. Le decisioni sono necessarie solo quando ci si avvicina alle caselle "intersezione", che sono indicate in verde nel diagramma. Quando è necessaria una decisione su quale direzione girare, la scelta viene effettuata in base a quale tessera adiacente all'intersezione porterà il fantasma più vicino alla sua casella target, misurata in linea retta. Viene misurata la distanza tra ogni possibilità e la tessera di destinazione e verrà selezionata la tessera più vicina all'obiettivo. Nell'immagine 2.3, il fantasma girerà verso l'alto all'incrocio. Se due o più scelte potenziali sono alla stessa distanza dall'obiettivo, la decisione tra di esse viene presa nell'ordine su→sinistra→giù. La decisione di uscire da destra non può mai essere presa in una situazione in cui due caselle sono equidistanti dal bersaglio, poiché qualsiasi altra opzione ha una priorità più alta. Poiché l'unica considerazione è quale tessera posizionerà immediatamente il fantasma più vicino al suo bersaglio, questo può far sì che i fantasmi selezionino la svolta "sbagliata" quando la scelta iniziale li avvicina, ma il percorso complessivo è più lungo. Un esempio è mostrato nell'immagine 2.4, dove la misurazione in linea retta fa sembrare l'uscita da sinistra una scelta migliore. Tuttavia, questo si tradurrà in una lunghezza complessiva del percorso di 26 caselle per raggiungere il bersaglio, quando l'uscita da destra avrebbe avuto un percorso lungo solo 8 tessere. Un ultimo caso speciale di cui tenere conto sono le quattro intersezioni colorate in giallo nell'immagine 2.1. Questi incroci specifici hanno una restrizione extra: i fantasmi non possono scegliere di girare verso l'alto da queste caselle. Se entrano da destra o da sinistra usciranno sempre dal lato opposto (salvo inversione di direzione forzata). Nota che questa restrizione non si applica alla modalità Frightened, e qui i fantasmi spaventati possono tornare in alto se la decisione avviene casualmente. Un fantasma che entra in queste tessere dall'alto può anche invertire la direzione verso l'alto se si verifica un cambio di modalità mentre entrano nella tessera, essendo che la restrizione viene applicata solo durante il processo decisionale "normale". Se Pac-Man viene inseguito da vicino dai fantasmi, può guadagnare terreno su di loro facendo una svolta verso l'alto in uno di questi incroci, poiché saranno costretti a prendere un percorso più lungo.

2.1.5 Modalità Scatter

Ogni fantasma ha una casella target predefinita e fissa mentre è in questa modalità, situata appena fuori dagli angoli del labirinto. Quando inizia la modalità Scatter, ogni

fantasma si dirigerà verso il proprio angolo di "casa" usando i normali metodi di ricerca del percorso. Tuttavia, poiché le caselle target effettive sono inaccessibili e i fantasmi non possono smettere di muoversi o invertire la direzione, sono costretti a continuare oltre il bersaglio, ma torneranno indietro il prima possibile. Ciò fa sì che il percorso di ogni fantasma alla fine diventi un anello fisso nel loro angolo. Se lasciato in modalità Scatter, ogni fantasma rimarrebbe nel suo ciclo indefinitamente. In pratica, la durata della modalità Scatter è sempre piuttosto breve, quindi i fantasmi spesso non hanno il tempo nemmeno di raggiungere il loro angolo o completare un circuito del loro loop prima di tornare alla modalità Chase. L'immagine 2.2 mostra la tessera bersaglio di ogni fantasma e l'eventuale percorso di loop, codificati a colori per abbinare il proprio colore.

2.2 I fantasmi

Figura 2.5: Fantasmi presenti all'interno di Pac-Man, con relativi nomi e nickname

Come è stato accennato in precedenza, le uniche differenze tra i fantasmi sono i loro metodi di selezione delle caselle target nelle modalità Chase e Scatter. L'unica descrizione ufficiale della personalità di ogni fantasma viene dalla descrizione di una sola parola "personaggio" mostrata nella modalità di attrazione del gioco. Daremo prima uno sguardo a come si comportano i fantasmi in modalità Scatter, poiché è estremamente semplice, quindi esamineremo l'approccio di ogni fantasma al targeting in modalità Chase.

2.2.1 Il fantasma rosso

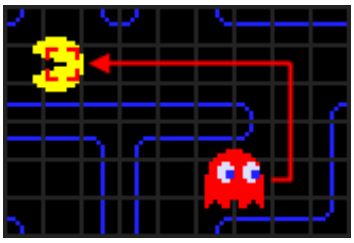

Figura 2.6: Rappresentazione della casella target del fantasma rosso

Figura 2.7: Design originale del fantasma rosso

Il fantasma rosso inizia fuori dalla casa fantasma, e di solito è il primo ad essere visto come una minaccia, dal momento che si dirige verso Pac-Man quasi immediatamente. Viene chiamato Blinky (nella nomenclatura moderna Clyde) e il gioco descrive la sua personalità come un'ombra (In giapponese, la sua personalità è indicata come "inseguitore"). Le descrizioni di entrambe le lingue sono accurate, poiché la casella target di Blinky in modalità Inseguimento è definita come la casella corrente di Pac-Man. Ciò assicura che Blinky segua quasi sempre direttamente dietro Pac-Man, a meno che il processo decisionale miope non lo induca a prendere una strada inefficiente. Anche se il metodo di mira di Blinky è molto semplice, ha una particolarità che gli altri fantasmi non hanno; in due punti definiti in ogni livello (in base al numero di punti rimanenti), la sua velocità aumenta del 5% e il suo comportamento cambia in modalità Scatter. La tempistica del cambio di velocità varia in base al livello, con il cambiamento che si verifica sempre prima man mano che il giocatore avanza. Il cambio alla modalità Scatter è forse più significativo dell'aumento della velocità, poiché fa sì che la casella bersaglio di Blinky rimanga nella posizione di Pac-Man anche in modalità Scatter, invece della sua casella fissa normale nell'angolo in alto a destra. Ciò mantiene Blinky in modo permanente in modalità Chase, anche se sarà comunque costretto a invertire la direzione a causa di un cambio di modalità. Quando si trova in questo stato avanzato, Blinky viene generalmente chiamato Cruise Elroy. Se Pac-Man muore mentre Blinky è in modalità Cruise Elroy, ritorna temporaneamente al comportamento normale, ma torna in modalità Elroy non appena tutti gli altri fantasmi sono usciti dalla zona di spawn.

2.2.2 Il fantasma rosa

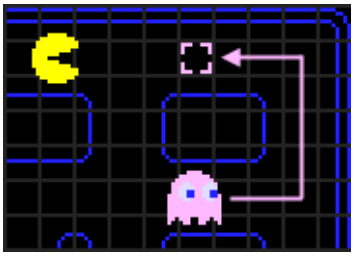

Figura 2.8: Rappresentazione della casella target del fantasma rosa

Figura 2.9: Design originale del fantasma rosso

Il fantasma rosa inizia all'interno della casa fantasma, ma esce sempre immediatamente, anche nel primo livello. Il suo soprannome è Pinky e la sua personalità è descritta come veloce. Questo è un notevole allontanamento dalla sua descrizione della personalità giapponese, "ambusher". La versione giapponese è molto più appropriata, dal momento che Pinky non si muove più velocemente di nessuno degli altri fantasmi (e più lentamente di Blinky in modalità Cruise Elroy), ma il suo schema di targeting tenta invece di spostarlo dove sta andando Pac-Man. di dove si trova attualmente. La casella target di Pinky in modalità Inseguimento è determinata osservando la posizione e l'orientamento attuali di Pac-Man e selezionando la posizione quattro caselle davanti a Pac-Man.

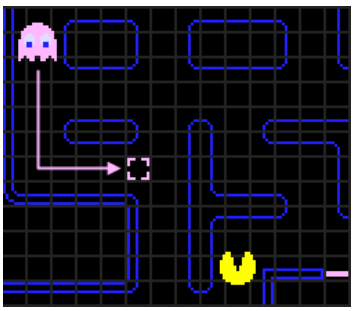

Figura 2.10: Esempio di errore di calcolo della casella target da parte del fantasma rosa

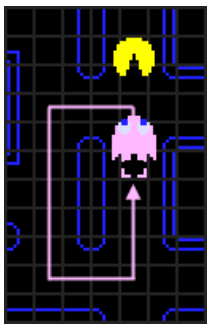

Figura 2.11: Esempio di bug del fantasma rosa

Almeno, questa era l'intenzione, e funziona quando Pac-Man è rivolto a sinistra, in basso o a destra, ma quando Pac-Man è rivolto verso l'alto, un errore di overflow nel codice del gioco fa sì che la casella target di Pinky venga effettivamente impostata

come quattro caselle davanti a Pac-Man e quattro caselle alla sua sinistra. Un'importante implicazione del metodo di targeting di Pinky è che Pac-Man può spesso vincere una partita di "pollo"[1] con lui. Poiché la sua casella target è posizionata quattro caselle davanti a Pac-Man, se Pac-Man si dirige direttamente verso di lui, la casella target di Pinky sarà effettivamente dietro di lui una volta che sono a meno di quattro tessere l'una dall'altra. Questo farà sì che Pinky scelga di prendere qualsiasi svolta disponibile per tornare indietro al suo obiettivo. Per questo motivo, è una strategia comune "fingere" momentaneamente di tornare nei confronti di Pinky se inizia a seguirlo da vicino. Questo spesso lo manderà in una direzione completamente diversa.

2.2.3 Il fantasma celeste

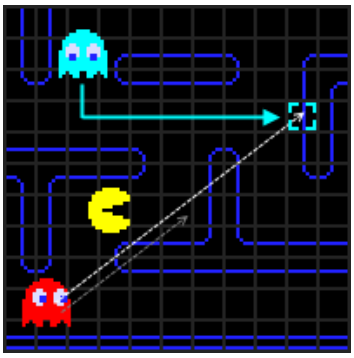

Figura 2.12: Rappresentazione della casella target del fantasma celeste

Figura 2.13: Design originale del fantasma celeste

Il fantasma celeste è soprannominato Inky e rimane all'interno della zona di spawn per un breve periodo al primo livello, non unendosi all'inseguimento finché Pac-Man non è riuscito a consumare almeno 30 dei punti. La sua descrizione della personalità in inglese è timida, mentre in giapponese è indicato come "capriccioso". Inky è difficile da prevedere, perché è l'unico dei fantasmi che usa un fattore diverso dalla posizione/orientamento di Pac-Man per determinare la sua casella target. Inky utilizza effettivamente sia la posizione/fronte di Pac-Man che la posizione di Blinky (il fantasma rosso) nei suoi calcoli. Per individuare l'obiettivo di Inky, iniziamo selezionando la posizione di due caselle davanti a Pac-Man nella sua attuale direzione di viaggio, in modo simile al metodo di mira di Pinky. Da lì, si immagina di disegnare

[1]Riferimento al Gioco del Pollo, una configurazione della teoria dei giochi a somma non nulla, in cui l'informazione è completa e vi partecipano due giocatori che agiscono contemporaneamente.

un vettore dalla posizione di Blinky a questa casella, quindi raddoppiare la lunghezza del vettore. La casella su cui finisce questo nuovo vettore esteso sarà il vero bersaglio di Inky. Di conseguenza, l'obiettivo di Inky può variare selvaggiamente quando Blinky non è vicino a Pac-Man, ma se Blinky è alle calcagna, lo sarà anche Inky in generale. Nota che il calcolo "due caselle davanti a Pac-Man" di Inky soffre esattamente dello stesso errore di overflow dell'equivalente a quattro tessere di Pinky, quindi se Pac-Man si sta dirigendo verso l'alto, il punto finale del vettore iniziale di Blinky (prima del raddoppio) lo farà in realtà ci sono due caselle in alto e due a sinistra di Pac-Man.

2.2.4 Il fantasma arancione

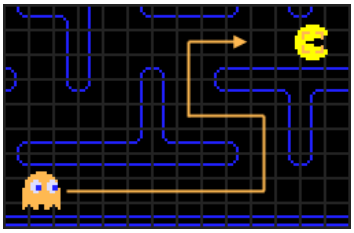

Figura 2.14: Rappresentazione della casella target del fantasma arancione (se la distanza tra lui e Pac-Man è maggiore di 8 caselle)

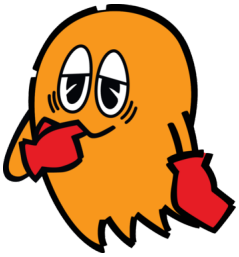

Figura 2.15: Design originale del fantasma arancione

Il fantasma arancione, Clyde (Blinky nella nomenclatura moderna), è l'ultimo a lasciare la zona di spawn e non esce affatto nel primo livello fino a quando più di un terzo dei punti non è stato mangiato. La descrizione della personalità inglese di Clyde è meschina, mentre la descrizione giapponese è "finta ignoranza". Come è tipico, la versione giapponese è più accurata, poiché il metodo di targeting di Clyde può dare l'impressione che stia solo "facendo le sue cose", senza preoccuparsi affatto di Pac-Man. La caratteristica unica del targeting di Clyde è che ha due modalità separate tra le quali passa costantemente avanti e indietro, in base alla sua vicinanza a Pac-Man. Ogni volta che Clyde ha bisogno di determinare la sua casella target, calcola prima la sua distanza da Pac-Man. Se è più lontano di otto caselle, il suo bersaglio è identico a quello di Blinky, usando la casella attuale di Pac-Man come target.

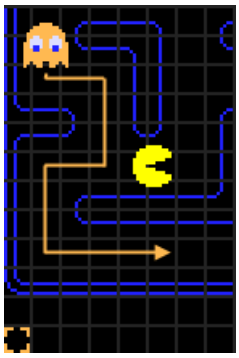

Figura 2.16: Rappresentazione della casella target del fantasma arancione (se la distanza tra lui e Pac-Man è minore di 8 caselle)

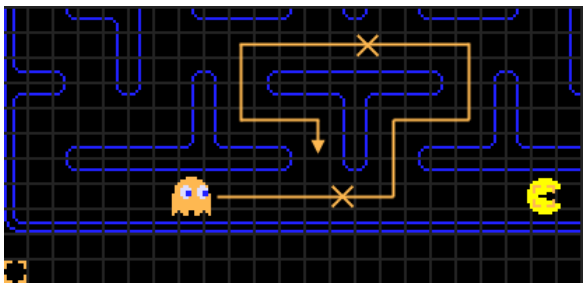

Figura 2.17: Esempio di cambio di modalità del fantasma arancione

Tuttavia, non appena la sua distanza da Pac-Man diventa inferiore a otto caselle, il bersaglio di Clyde viene impostato sulla stessa casella fissa in modalità Scatter, appena fuori dall'angolo in basso a sinistra del labirinto. La combinazione di questi due metodi ha l'effetto complessivo di Clyde che alterna il venire direttamente verso Pac-Man, e poi cambiare idea e tornare al suo angolo ogni volta che si avvicina troppo. Nell'immagine 2.17, i segni X sul percorso rappresentano i punti in cui la modalità di Clyde cambia. Se Pac-Man riuscisse in qualche modo a rimanere fermo in quella posizione, Clyde girerebbe indefinitamente intorno a quell'area a forma di T. Fintanto che il giocatore non è nell'angolo inferiore sinistro del labirinto, Clyde può essere evitato completamente assicurandosi semplicemente di non bloccare la sua "via di fuga" al suo angolo. Mentre Pac-Man si trova entro otto caselle dall'angolo inferiore sinistro, il percorso di Clyde finirà esattamente nello stesso ciclo che avrebbe alla fine mantenuto in modalità Scatter.

Capitolo 3

Un'implementazione di base

Ricreare per intero Pac-Man in un qualsiasi linguaggi di programmazione richiede un quantitativo di tempo elevato. Pertanto l'idea migliore è quella di prendere una versione già esistente, e modificarla/correggerla, in modo da ricreare una versione del gioco il più simile possibile alla versione originale. Va detto che molte versioni di Pac-Man, ricreate in diversi linguaggi di programmazione, presentano veri problemi:

- Discrepanze tra le meccaniche originali e quelle implementate;
- Errori di realizzazione;
- Mancanza di elementi essenziali;
- Codice mal realizzato.

Pertanto, un'opzione ottimale è quella di scegliere una versione di Pac-Man, fissato un certo linguaggio, che non risulti "troppo distante" dalla versione originale del gioco. Per questa tesi, come linguaggio utilizzato, si è scelto il Python, poichè si adatta alla realizzazione di videogiochi, ma soprattutto per la possibilità di realizzare in modo semplice agenti in grado di apprendere.

3.1 Prima di iniziare...

Prima di lavorare sul codice, è necessario installare due librerie:

- `numpy`: è una libreria che permette di lavorare in modo più agevolato con vettori e matrici multidimensionali;
- `pygames`: è una libreria che permette di lavorare con vettori e interfacce 2D.

Bisogna inoltre spiegare alcuni aspetti fondamentali che verranno usati:

- Per rappresentare la mappa di gioco, verrà usata una griglia $28 * 31$, con caselle grandi 8 px;
- Ogni casella è rappresentata dalle coordinate (x, y);
- L'ordine di lettura delle righe è da sopra verso sotto, mentre quello delle colonne è da sinistra verso destra;
- Per i nomi dei fantasmi verrà usata la nomenclatura moderna (rosso: Clyde; rosa: Pinky; celeste: Inky; arancione: Blinky).

3.2 Descrizione qualitativa del codice trovato

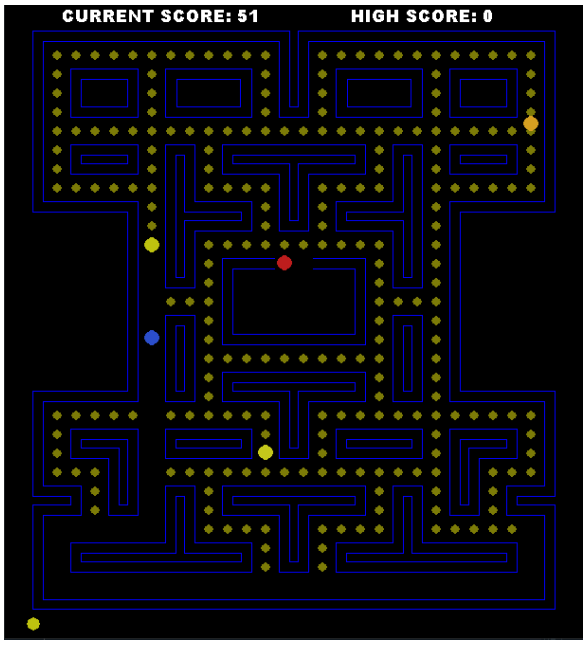

Figura 3.1: Una schermata di gioco dell'implementazione di base.

La versione che si è scelta per questo progetto è quella di *A Plus Coding*. Tra tutte le versioni esaminate, questa è una delle poche che presenta una struttura del gioco abbastanza ben definita, seppur sono presenti dei bug/discrepanze con la versione originale del gioco. Un dettaglio interesante è che la seguente è una delle poche versioni che non solo implementa una mappa molto simile a quella originale del gioco, ma implementa anche tutti e quattro i fantasmi. Molte versioni di Pac-Man che

sono state realizzate e pubblicate su internet, infatto, usano una mappa di gioco più piccola dell'originale, implementando solo due dei qattro fantasmi. Nell'appendice A è possibile vedere una copia del codice originale da cui si è partiti, inoltre è possibile scaricare tale versione tramite la *repository* di GitHub dove è stato pubblicato.

3.3 Problemi dell'implementazione di base

Questa versione, come preannunciato prima, presenta molti errori/discrepanze rispetto al gioco originale:

- La colorazione dei fantasmi errata;
- La velocità di Pac-Man e dei fantasmi errata;
- Il behaviour dei fantasmi diverso da quello originale;
- La mancanza dei due corridoi laterali;
- L'errata rappresentazione dei dots;
- Mancata implementazione dei pellet, con relativa mancanza delle modalità "Frightened" ed "Eaten" dei fantasmi;
- Stile grafico mal realizzato;
- Errori vari presenti nel codice;
- Mancanza completa di commenti.

Capitolo 4

La revisione dell'implementaizone di base: primi risultati

COme discusso prima, l'implementazione di base scelta presenta molte incongruenze con la versione originale del gioco. È pertanto necessario apportare importanti modifiche.

4.1 Modifiche apportate

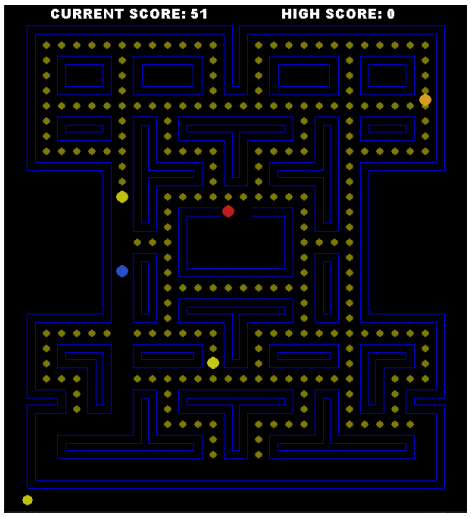

Figura 4.1: Una schermata dell'implementazione di base del gioco.

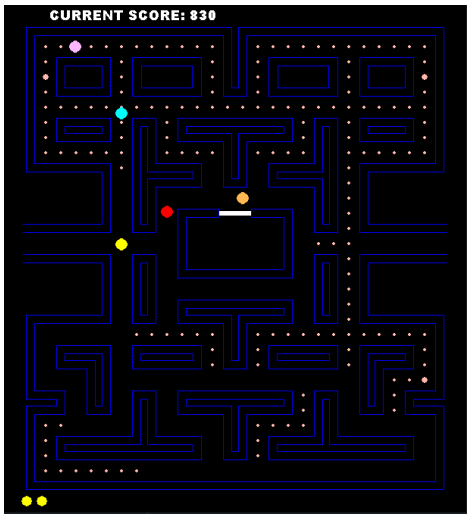

Figura 4.2: Una schermata dell'implementazione rivisitata del gioco.

La seguente è una versione di Pac-Man ricreato in Python, partendo dalla versione presentata prima. Viene utilizzata la nomenclatura moderna per i fantasmi, pertanto

il fantasma rosso viene chiamato Clyde, mentre quello arancione è chiamato Blinky. Le modifiche principale apportate riguardano il comportamento dei fantasmi e le meccaniche core del gioco; inoltre sono state apportate delle migliorie grafiche, in modo da renderlo il più simile possibile al gioco originale. Nella realizzazione di questa versione non sono trattati alcuni elementi:

- La modalità "Scatter" dei fantasmi;
- La modalità "Cruise Elroy" di Clyde;
- L'apparizione della frutta nel labirinto.

Inoltre tutti gli incroci presenti sulla mappa vengono considerati dai fantasmi per la scelta della direzione successiva. Il codice, che viene presentato nell'appendice B, è inoltre compreso di commenti per spiegare in dettaglio ciò che viene fatto in ogni singola funzione.

4.2 Aspetti principali del codice modificato

Di seguito sono spiegate nel dettaglio le modifiche principale apportate, con relativa spiegazione dell'implementazione.

4.2.1 Modifica del comportamento dei fantasmi

La prima modifica importante al codice che è stata apportata è la modifica dei comportamenti dei fantasmi: sono state cambiate sia la scelta delle caselle target, sia della scelta della mossa successiva, sia la velocità di movimento dei fantasmi. Non sono state implementate la modalità "Scatter" dei fantasmi, sia la modalità "Cruise Elroy". Un dettaglio importante che è Stato implementato è la scelta della mossa: in caso di due mosse con distanza minima uguale, viene seguita la sequenza *Sopra* $\rightarrow$ *Sinistra* $\rightarrow$ *Sotto* $\rightarrow$ *Destra* (il fantasma sceglierà di muoversi a destra sempre e solo se tale direzione gli permette di decrementare la distanza tra lui e il player).

4.2.2 Implementazione dei pellet e delle modalità "Frightened" ed "Eaten"

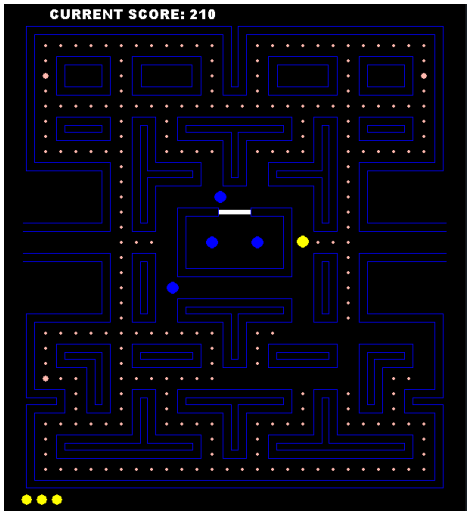

Figura 4.3: Un esempio di fantasmi nello stato di "Frightened" durante una partita.

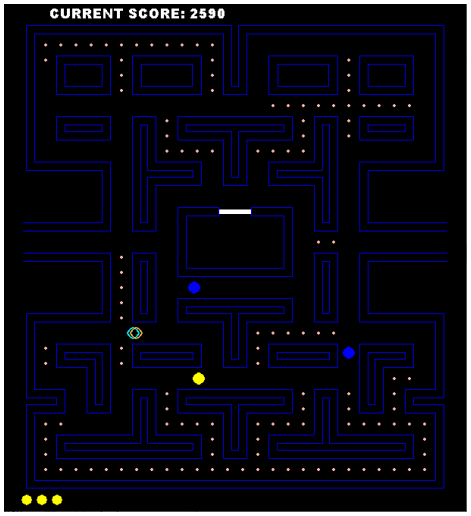

Figura 4.4: Un esempio di fantasmi nello stato di "Eaten" durante una partita.

Una seconda modifica importante è l'implementazione dei pellet, con la relativa implementazione nei fantasmi della modalità "Frightened". È stato sia implementato un timer di 10 secondi che viene attivato/resettato ogni volta che Pac-Man mangia un pellet. Durante questo lasso di tempo viene modificato il colore dei fantasmi. Nel caso in cuil il fantasma viene mangiato da Pac-man, il fantasma entra nello stato di "Eaten", e la sua casella target diventa il centro della zona di spawn. A livello grafico, il fantasma viene renderizzato come un cerchio vuoto.

4.2.3 Implementazione delle vite

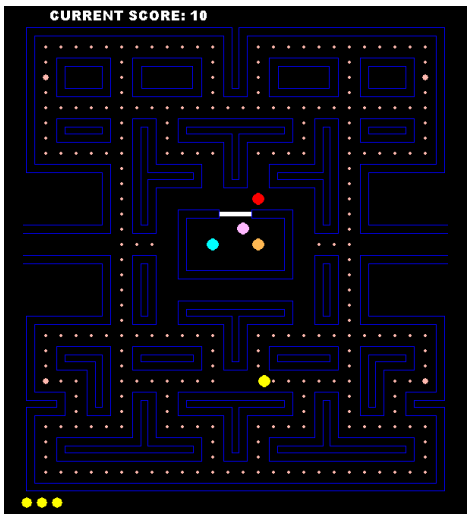

Figura 4.5: La schermata di gioco appena viene iniziata una partita.

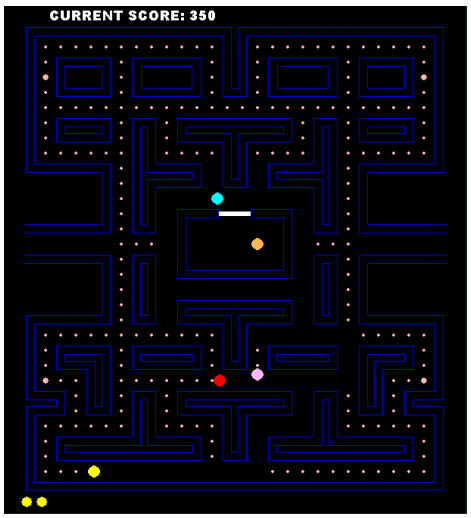

Figura 4.6: La schermta di gioco dopo aver perso la partita (il numero di "player" in basso a sinistra è decrementato di 1).

Un'aggiunta abbastanza interessante, seppur non troppo complessa, è l'aggiunta delle vite e delle relative meccaniche: ogni volta che il player incontra un fantasma che è in modalità "Scatter" o "Chase", viene in primis decrementato il contatore delle vite; nel caso tale contatore risulta uguale a zero, il gioco passa in modalità "Game over", mostrando la relativa schermata; in caso contrario, viene resettato posizione e modalità dei fantasmi, oltre che la posizione del player.

Capitolo 5

Pac-Man come agente intelligente: l'agente player

Realizzata una versione del gioco il pìu simile possibile versione originale, possiamo realizzare l'agente che controllerà il player, e userà lo stato dell'ambiente per decidere quale azione effettuare.

5.1 L'ambiente

È importante definire alcuni punti chiave:

- Il nostro ambiente è rappresentabile tramite un *sistema dinamico*, essendo in grado di mutare nel tempo con un certo grado di libertà;
- Lo stato dell'ambiente, essendo *completamente osservabile*, è rappresentabile tramite un array contenente le informazioni riguardo il player, i fantasmi, la posizione di dot/pellet e muri, e il numero di vite rimanenti;
- Oltre ad essere un sistema dinamico, il nostro ambiente è possibile definirlo come un *processo aleatorio*: ciò è dato dalla modalità "Frightened", durante la quale i fantasmi scelgono casualmente la direzione successiva ogni volta che raggiungono un incrocio (l'unico vincolo imposto è che non possono scegliere la direzione da cui provengono);
- Poiché la probabilità di finire in un certo stato successivo s_{t+1} dipende solo dallo stato s_t e dall'azione a_t che viene eseguita dallo stato attuale, è possibile dire che il nostro ambiente, essendo rappresentabile da un processo aleatorio, è definibile

tramite un *processo Markoviano*

$$P(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, ..., s_1, a_1, s_0, a_0) = P(s_{t+1} \mid s_t, a_t)$$

È possibile usare il codice presentato nell'appendice B come punto di partenza per la realizzazione dei nostri agenti, essendo non solo un'implementazione abbastanza realistica del gioco, ma rispecchiando inoltre le caratteristiche qui sopra citate.

5.2 La funzione di decisione

Il nostro agente basato su modello utilizza una funzione di utility basata sui Q-values:

$$U(s) = \max_a Q(s, a)$$

Ciò permette al nostro agente non necessitare un modello dell'ambiente, e ci permette di definire la policy dellágente nel seguente modo:

$$\pi(s) = \operatorname*{argmax}_a Q(s, a)$$

5.2.1 Proprietà osservabili necessarie alla decisione

Le proprietà osservabili che saranno necessarie sia alla decisione della mossa successiva, sia alla rappresentazione degli stati, sono le seguenti:

- Posizione (x, y) attuale del player;
- Direzione (δ_x, δ_y) attuale del player;
- Posizione $(x^{(f)}, y^{(f)})$ attuale di ogni fantasma;
- Direzione $(\delta_x^{(f)}, \delta_y^{(f)})$ attuale di ogni fantasma;
- Stato $\mu^{(f)}$ attuale di ogni fantasma;
- Numero di dot/pellet rimanenti;
- Posizione $(x^{(d)}, y^{(d)})$ di ognuno dei dot;
- Posizione di tutti i muri nella mappa;
- Numero di vite rimanenti;
- Punteggio attuale.

Dove f è l'identificativo dei fantasmi, mentre d è l'identificativo dei dot.

5.2.2 Statistiche sui valori assunti da quelle proprietà

Ognuna delle proprietà osservabili citate sopra possiede un proprio intervallo di valori che può assumere:

- Posizione x del player: $x \in [0, 27]$;
- Posizione y del player: $y \in [0, 30]$;
- Direzione (δ_x, δ_y) del player: $(\delta_x, \delta_y) \in \{(0, -1), (-1, 0), (0, 1), (1, 0)\}$
- Posizione $x^{(f)}$ di ogni fantasma: $x^{(f)} \in [0, 27]$;
- Posizione $y^{(f)}$ di ogni fantasma: $y^{(f)} \in [0, 30]$;
- Direzione $(\delta_x^{(f)}, \delta_y^{(f)})$ del player: $(\delta_x^{(f)}, \delta_y^{(f)}) \in \{(0, -1), (-1, 0), (0, 1), (1, 0)\}$
- Stato $\mu^{(f)}$ di goni fantasma:

$$\mu^{(f)} = \begin{cases} 1 \text{ se il fantasma è nella modalità “Chase”} \\ 0 \text{ se il fantasma è nella modalità “Eaten”} \\ -1 \text{ se il fantasma è nella modalità “Frightened”} \end{cases}$$

- Posizione x di un dot/pellet: $x^{(d)} \in [0, 27]$;
- Posizione y di un dot/pellet: $y^{(d)} \in [0, 30]$;
- La presenza o meno di muri in una certa posizione:

$$\omega(x, y) = \begin{cases} 1 \text{ se è presente un muro in quella direzione} \\ 0 \text{ altrimenti} \end{cases}$$

- Il numero di vite rimanenti: $l \in [0, 3]$
- Il numero di dot/pellet rimanenti: $r_d \in [0, 248]$.

5.3 Automazione dell'agente: codifica manuale

Una codifica manuale dell'agente realizzabile che permette all'agente di scegliere la mossa successiva è la seguente: usare una funzione $f : \mathbb{R} \to \mathbb{R}$ che utilizza le informazioni dello stato attuale. Definendo le seguenti quantità:

- $\delta^{(f)} = (x - x^{(f)})^2 + (y - y^{(f)})^2$ è il rapporto tra distanza attuale tra player e fantasma;
- $\delta^{(f)}_{(\delta_x,\delta_y)} = (x_{(\delta_x,\delta_y)} - \bar{x}^{(f)})^2 + (y_{(\delta_x,\delta_y)} - \bar{y}^{(f)})^2$ è il rapporto tra distanza successiva data una certa mossa tra player e fantasma;
- $\Delta_{(\delta_x,\delta_y)} = \sum_f \delta^{(f)}_{(\delta_x,\delta_y)}$ è la somma delle distanze tra le distanze successive data la mossa tra player e fantasmi;
- $\pi_{(\delta_x,\delta_y)} = \dfrac{(x - x^{(d)})^2 + (y - y^{(d)})^2}{(x_{(\delta_x,\delta_y)} - x^{(d)})^2 + (y_{(\delta_x,\delta_y)} - y^{(d)})^2 + 1}$ è il rapporto tra la distanza attuale e quella successiva alla mossa tra player e dot più vicino.

Dove $(\bar{x}^{(f)}, \bar{y}^{(f)})$ è la posizione sucessiva del fantasma, mentre $(x_{(\delta_x,\delta_y)}, y_{(\delta_x,\delta_y)})$ indica la posizione del player dopo la mossa (δ_x, δ_y), possiamo scrivere la formula usata per calcolare l'euristica di ogni mossa nell'algoritmo come:

$$\forall (\delta_x,\delta_y) \in \{(0,-1),(-1,0),(0,1),(1,0)\},$$
$$f(\delta_x,\delta_y) = \frac{\sum_f \mu^{(f)}(\delta^{(f)}_{(\delta_x,\delta_y)} - \delta^{(f)})(\Delta_{(\delta_x,\delta_y)} - \delta^{(f)}_{(\delta_x,\delta_y)})}{\Delta_{(\delta_x,\delta_y)}} + \pi_{(\delta_x,\delta_y)}$$

Questa formula può essere spiegata scomponendola ed analizzando i vari pezzi di cui è composta:

- $d^{(i)}_{(\delta_x,\delta_y)} - d^{(i)}$ assumerà segno positivo se la distanza tra agente e fantasma aumenta dopo aver effettuato la mossa, mentre avrà segno negativo in caso contrario;
- Più il fantasma è vicino all'agente, più $D_{(\delta_x,\delta_y)} - d^{(i)}_{(\delta_x,\delta_y)}$ è grande, e quindi il fantasma relativo avrà maggiore influenza;
- Dividere per $D_{(\delta_x,\delta_y)}$ permette di valutare tutti i valori proporzionati rispetto alle distanze tra agente e fantasmi che si avrebbero dopo una mossa dell'agente;
- Il valore $\pi_{(\delta_x,\delta_y)}$ permette di scegliere una mossa anche in base alla distanza tra l'agente e il dot più vicino, in modo che, se vengono trovate due buone mosse, è possibile decidere quale delle due porta l'agente ad avvicinarsi ad un dot (esattamente, mi permette di decidere quale mossa mi porta più vicino al dot con minore distanza dalla nuova posizione dell'agente);

- Nel calcolo di $\pi_{(\delta_x,\delta_y)}$ è presente un 1, questo per evitare che il denominatore diventi uguale a 0, nel caso in cui una mossa porti l'agente a finire direttamente su un dot;

Ovviamente, è necessario calcolare il valore di questa formula per ognuna delle possibili direzioni che l'agente può prendere, e identificare come direzione successiva quella con associato il valore della funzione più alto. Da qui nasce però un problema: nel caso in cui l'agente non possa muoversi in una certa direzione, il calcolo di questo valore non deve essere effettuato, o per essere più precisi, il valore da considerare per quella mossa deve essere il più basso possibile. Si può usare un escamotage molto semplice per risolvere tale problema, ossia si utilizza una "funzione dummy" che si comporta nel seguente modo:

$$\bar{f}(\delta_x, \delta_y) = \begin{cases} f(\delta_x, \delta_y) \text{ se l'agente si può muovere nella direzione } (\delta_x, \delta_y) \\ -100000 \text{ altrimenti} \end{cases}$$

Calcolate le euristiche per le varie mosse possibili, la mossa sucessiva viene calcolata tramite la seguente policy:

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, \bar{f}(\delta_x, \delta_y)$$

Nell'appendice C viene riportata una possibile implementazione di questo agente programmato.

5.4 Automazione dell'agente: adattamento tramite reinforcement

Questo agente programmato, seppur presenta delle caratteristiche interessanti, non è in grado di completare con successo una partita a Pac-Man. Ciò dericva dal fatto che la funzione usata non permette di prevedere con anticipo le mosse successive che i fantasmi eseguiranno date tutte le mosse che l'agente può eseguire in un lasso di tempo k. L'utilizzo del reinforcement learning può risolvere questo problema: tramite l'utilizzo di reward e rinforzi, fornti in base al risultato delle azioni dell'agente, esso è in grado di imparare non solo cosa deve fare, ma anche come funziona l'ambiente in cui è posto.

5.4.1 Funzione e parametri del modello

Il nostro agente basato su modello utilizza la funzione di decisione citata sopra per la scelta della mossa successiva:

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, Q_\phi(s, a)$$

Ad ogni azione possibile viene assegnata un Q-value predetto da una Deep Q-network (DQN), che si occupa di prevedere i Q-values. La DQN utilizzata è formata nel seguente modo:

- 1 layer di input composto da 14 neuroni
- 2 layer nascosti composti da 50 neuroni;
- 1 layer di output composto da 4 neuroni.

I valori di input che vengono passati alla nostra DQN sono:

- La posizione (x, y) attuale del player;
- Lo stato attuale di ogni singolo fantasma, definito da posizione e stato tramite la seguente funzione:

$$\sigma^{(f)} = 10^{1-\mu^{(f)}} \cdot (|x - x^{(f)}| + |y - y^{(f)}|)$$

- La posizione (x, y) del dot più vicino al player;
- La presenza o meno di muri nelle quattro direzioni intorno al player;
- Il numero di vite rimanenti;
- Il numero di dot/pellet rimanenti.

La funzione di attivazione usata è la funzione di rettificazione $f(x) = x^+ = \max(0, x)$, la quale viene solo applicata sui 2 layer nascosti. Inoltre, la nostra DQN viene addestrata tramite discesa del gradiente, la quale usa la L_2 loss function, applicata sul batch estratto dalla replay memory (s_t, a_t, s_{t+1}, r_t) di egual grandezza della memoria (k tuple salvate nella replay memory, k tuple contenute nel batch):

$$\phi \leftarrow \phi - \alpha \frac{1}{k} \sum_i \frac{d}{d\phi} (Q_\phi(s_i, a_i) - [r_i + \gamma \max_a Q_\phi(s_{i+1}, a)])^2$$

Tutte le tuple (*stato*, *azione*, *nuovo stato*, *reward*) vengono memorizzate in una replay memory, la cui grandezza varia con il numero di partite effettuate. Questa memoria ha una struttura ciclica, e La funzione di reward assegna una specifica reward ad ogni evento importante che può verificarsi all'interno di una partita, come indicato nella tabella sottostante:

Valore	**Descrizione degli eventi associati**
-20	L'agente non si può muovere in quella direzione.
10	L'agente ha mangiato un dot/pellet.
50	L'agente ha mangiato un fantasma.
-50	L'agente ha perso una vita.
-500	L'agente ha perso la partita.
100	L'agente ha vinto la partita.
-1	L'agente effettua una mossa.

Questa reward è cumulativa (se avvengono diversi eventi, le loro rispettive reward vengono sommate), e la reward ottenuta in ogni stato viene propagata negli stati successivi, con un decadimento progressivo (maggiore il numero di partite effettaute, maggiore la "durata" di una reward). Alla reward di ogni tupla (*stato*, *azione*, *nuovo stato*, *reward*) viene aggiunto inoltre un aggiustamento calcolato tramite la distanza tra player e i fantasmi tramite la seguente funzione: Sia $\delta_t^{(f)} = |x^{(f)} - x| + |y^{(f)} - y|$ la distanza tra player e il fantasma f al tempo t, sia $\Delta_t^{(f)} = \delta_t^{(f)} - \delta_{t-1}^{(f)}$ la differenza tra la distanza attuale e quella precedente tra player e il fantasma f, sia σ_{f_t} lo stato al tempo t del fantasma, abbiamo che il nostro aggiustamento è uguale a

$$\rho_t^{(f)} = \mu_t^{(f)} \cdot \max\{e^{6-\delta_t^{(f)}} - 1, 0\} \cdot \text{sign}\,(\Delta_t^{(f)}) \cdot e^{-\Delta_t^{(f)}}$$

5.4.2 Algoritmo in pseudocodice

Di seguito viene riportato l'algoritmo di training in pseudocodice. È importante notare che la replay memory contiene un numero preciso di tuple (*stato*, *azione*, *nuovo stato*, *reward*), e quando viene salvata una nuova tupla nella memoria viene eliminata la tupla più vecchia che è stata inserita. Inoltre il parametro EPOCHS viene usato per indicare quante partite vengono effettuate (un'epoca comprende 5 partite).

Algorithm 1 Deep Q-learning con Replay memory

procedure DQN-Learning(CYCLES, EPS_DECAY, EPS_DECREASE, EPS˙MIN)
    Inizializza la rete neurale Q_ϕ con pesi casuali
    for cycle $= 1$ **to** CYCLES $* 5$ **do**
        Inizializza la replay memory R con capacità $n = 5 + \lfloor(\log_5 n) * (\log_{10} n)\rfloor$
        Inizializza EPSILON $= 1 - (1 - \text{cycle}) *$ EPS_DECREASE
        while il player vince o perde la partita **do**
            Genera lo stato attuale s_t
            Genera un numero randomico x tra 0 ed 1
            if $x \leq$ EPSILON **then**
                Sceglie un'azione randomica a_t
            else
                Sceglie $a_t = \max_a Q_\phi(s_t, a)$
            Esegue l'azione a_t e calcola la reward r_t
            Genera il nuovo stato attuale s_{t+1}
            Salva la transizione (s_t, a_t, s_{t+1}, r_t) in R
            if la memoria R è stata completamente aggionata **then**
                Estrae il batch delle transizioni (s_t, a_t, s_{t+1}, r_t) salvato in memoria (k transizioni)
                Calcola $y_t = Q_\phi(s_t, a_t)$
                Calcola $p_t = r_t + \gamma * \max_a Q_\phi(s_{t+1}, a)$
                Effettua uno step della discesa del gradiente su $L_2(p_t, y_t) = (p_t - y_t)^2$
                EPSILON $= \min\{\text{EPSILON} - \text{EPS˙DECAY}, \text{EPS˙MIN}\}$

5.4.3 Misure prestazionali

Partiamo dal definire un agente completamente randomico, ossia che non osserva lo stato attuale dell'ambiente e sceglie con una probabilità equa quale mossa effettuare; il nostro agente verrà addestrato per un periodo di k epoche, dove ogni epoca corrisponde a 5 partite. Durante il periodo di training, osserveremo tre valori:

- Il punteggio medio, o score medio, effettuato durante un'epoca;
- Il numero medio di mosse effettuate durante un'epoca (si considera mossa lo spostamento da una casella ad un'altra, pertanto sono esclusi i movimenti in direzione di un muro).
- Il punteggio medio per mossa effettauto durante un'epoca.

Capitolo 6

Valutazione delle prestazioni

Per effettuare una valutazione del nostro agente, dopo aver effettuato un training di 500 epoche (ricordiamo che un'epoca comprende 5 partite), possiamo comparare i valori medi di score, numero di mosse e punteggio medio per mossa, per ogni singola epoca, con quelli del nostro agente programmato e di un agente puramente randomico, ossia un agente che decide randomicamente quale mossa effettuare, senza osservare lo stato in cui si trova. I valori mostrati riguardano nello specifico le prime 250 epoche di tranining.

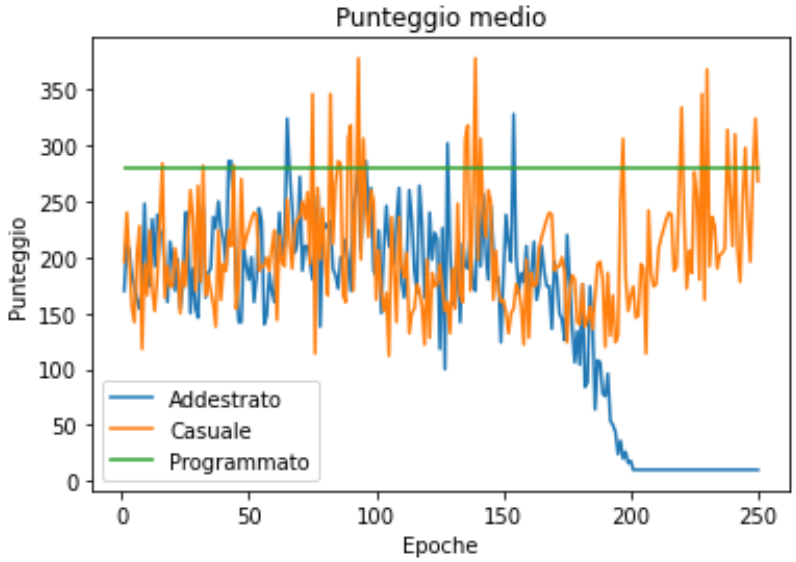

Figura 6.1: Grafico dell'andamento dello score medio nelle varie epoche.

In questo primo grafico viene mostrato il variare dello scure medio durante tutte le epoche di training dell'agente. Come possiamo vedere, seppur nelle prime 150 epoche

dimostra un buon andamento, dalla 150-esima epoca fino alla 200-esima epoca avviene una veloce discesa, dovuta forse ad un'involuzione dell'agente, non riuscendo quindi ad apprendere come guadagnare punti durante la partita.

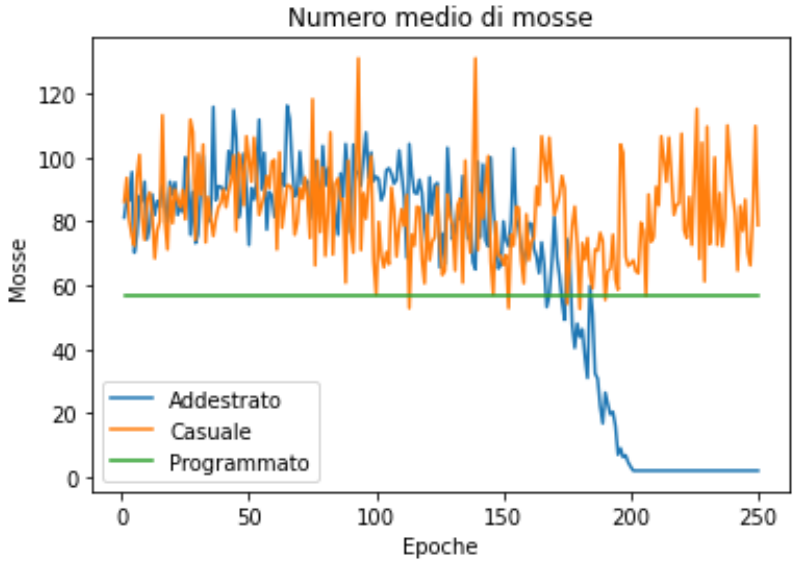

Figura 6.2: Grafico dell'andamento del numero medio di mosse nelle varie epoche.

In questo secondo grafico, invece, ci permette di controntare il numero medio di mossa fatte sia dal nostro agente in grado di apprendere, sia dall'agente randomico. Anche qui, come nel primo grafico, seppur nelle prime 150 epoche il nostro agente semobra dimostrare alcune capacità, dopo la 150-esima inizia un'involuzione, raggiungendo alla 200-esima epoca un punto di stallo e non riuscendo ad effettuare più mosse.

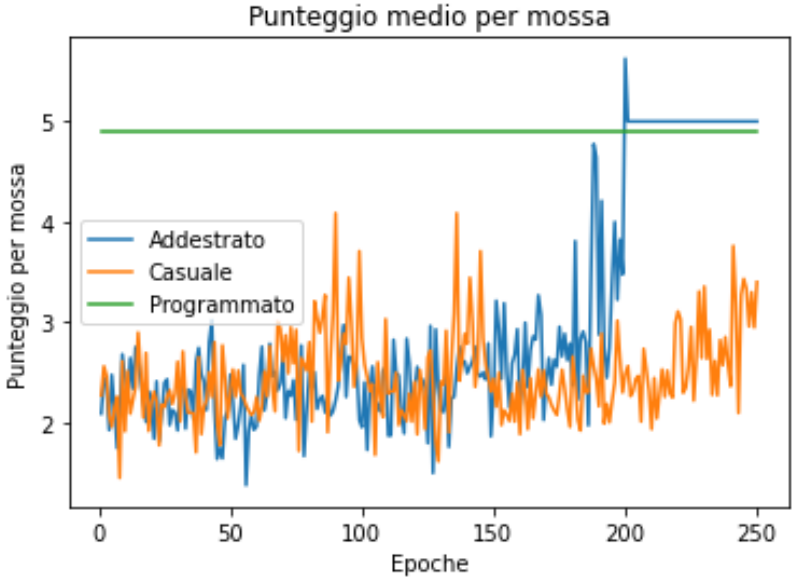

Figura 6.3: Grafico dell'andamento del numero medio di vite perso nelle varie epoche.

In questo terzo e ultimo grafico viene mostrato il confronto tra il punteggio medio per mossa durante tutte le epoche. Un dettaglio interessante che possiamo notare è che, seppur il nostro agente sembra entrare in una fase di involuzione, presenta un punteggio medio per mossa alto rispetto ad un agente completamente randomico. Possiamo ipotizzare quindi che, seppur non riesce a concludere con successo una partita, riesce ad ottimizzare nel miglio modo possibile le poche mosse che riesce a fare. Considerando tutti i dati raccolti, possiamo concludere che il nostro agente non risulta essere in grado di apprendere una funzione action-utility. In particolare, possiamo dire che il nostro agente non riesce ad imparare delle tecniche per evitare i fantasmi e i muri. Osservando i risultati ottenuti, le considerazioni effettuate, e le due implementazioni mostrate, possiamo concludere che le due soluzioni proposte, seppur con dettagli interessanti e soluzioni peculiari, non sono in grado di completare con successo una partita a Pac-Man. Ciò può dipendere da due fattori:

- Il nostro agente programmato, seppur è in grado di pesare e definire delle priorità sulle proprietà osservabili del nostro ambiente, non è in grado di prevedere le azioni future dei fantasmi;
- Al contrario, seppur il nostro agente è attrezzato in modo da prevedere teoricamente in leggero anticipo le mosse dei fantasmi, non riesce a muoversi corret-

tamente nell'ambiente, dovuto forse alla non capacità di assegnare una priorità alle proprietà osservabili.

Conclusione

È stato realizzato un agente artificiale che simula il gioco di Pac-Man, esplicitando lo stato dell'agente, le azioni possibili e la funzione di utilità. È stata definita una strategia di gioco programmata basata su euristiche. È stato progettato un sistema di reinforcement learning basato su rete neurale per simulare la strategia di gioco ed il comportamento dell'agente. La sperimentazione è in corso e i primi risultati manifestano dei limiti nella capacità dell'apprendimento per rinforzo nello sfuggire a configurazioni localmente utili che non consentono di procedere nel gioco (minimi locali).

6.1 Future evoluzioni

- L'utilizzo di una CNN (Convolutional Neural Network), ossia una rete neurale in grado di elaborare immagini, permetterebbe di utilizzare una foto dello stato attuale del gioco, considerando tutti gli elementi presenti durante la partita;
- Si può considerare l'implementazione di una double-DQN al posto di una DQN, ossia l'utilizzo di due DQN, usate l'una per il training dell'altra, poiché possiamo considerare il nostro ambiente come rumoroso, e in tali ambienti il Q-learning può talvolta sovrastimare i valori dell'azione, rallentando l'apprendimento.
- L'utilizzo di una Long Short-Term Memory (LSTM) piuttosto di una Replay Memory permettere teoricamente di rendere più stabile l'apprendimento, poiché sono molto adatte per problemi di classificazione, processare e effettuare predizioni basate su una serie storica di dati.

Appendice A

Codice dell'implementazione di base

Di seguito viene riportato il codice sorgente dell'implementazione.

A.1 File "walls.txt"

```
1111111111111111111111111111
1CCCCCCCCCCCC11CCCCCCCCCCCC1
1C1111C11111C11C11111C1111C1
1C1111C11111C11C11111C1111C1
1C1111C11111C11C11111C1111C1
1CCCCCCCCCCCCCCCCCCCCCCCCCC1
1C1111C11C11111111C11C1111C1
1C1111C11C11111111C11C1111C1
1CCCCCC11CCCC11CCCC11CCCCCC1
111111C11111C11C11111C111111
111111C11111C11C11111C111111
111111C11CCCCCCCCCC11C111111
111111C11C111BB111C11C111111
111111C11C15000041C11C111111
111111CCCC10000001CCCC111111
111111C11C12000031C11C111111
111111C11C11111111C11C111111
111111C11CCCCCCCCCC11C111111
111111C11C11111111C11C111111
111111C11C11111111C11C111111
1CCCCCCCCCCCC11CCCCCCCCCCC1
1C1111C11111C11C11111C1111C1
1C1111C11111C11C11111C1111C1
1CCC11CCCCCCCCCCCCCCCC11CCC1
111C11C11C11111111C11C11C111
111C11C11C11111111C11C11C111
1CCCCCC11CCCC11CCCC11CCCCCC1
1C1111111111C11C1111111111C1
1C1111111111C11C1111111111C1
1CCCCCCCCCCCCPCCCCCCCCCCCCC1
1111111111111111111111111111
```

A.2 File "settings.py"

```
from pygame.math import Vector2 as vec

# screen settings
WIDTH, HEIGHT = 610, 670
FPS = 60
TOP_BOTTOM_BUFFER = 50
MAZE_WIDTH, MAZE_HEIGHT = WIDTH-TOP_BOTTOM_BUFFER, HEIGHT-TOP_BOTTOM_BUFFER

ROWS = 30
COLS = 28

# colour settings
BLACK = (0, 0, 0)
RED = (208, 22, 22)
GREY = (107, 107, 107)
WHITE = (255, 255, 255)
PLAYER_COLOUR = (190, 194, 15)

# font settings
START_TEXT_SIZE = 16
START_FONT = 'arial black'

# player settings
# PLAYER_START_POS = vec(2, 2)

# mob settings
```

A.3 File "player_class.py"

```
import pygame
from settings import *
vec = pygame.math.Vector2


class Player:
    def __init__(self, app, pos):
        self.app = app
        self.starting_pos = [pos.x, pos.y]
        self.grid_pos = pos
        self.pix_pos = self.get_pix_pos()
        self.direction = vec(1, 0)
        self.stored_direction = None
        self.able_to_move = True
        self.current_score = 0
        self.speed = 2
        self.lives = 1

    def update(self):
        if self.able_to_move:
            self.pix_pos += self.direction*self.speed
        if self.time_to_move():
            if self.stored_direction != None:
                self.direction = self.stored_direction
            self.able_to_move = self.can_move()
        # Setting grid position in reference to pix pos
        self.grid_pos[0] = (self.pix_pos[0]-TOP_BOTTOM_BUFFER +
                            self.app.cell_width//2)//self.app.cell_width+1
        self.grid_pos[1] = (self.pix_pos[1]-TOP_BOTTOM_BUFFER +
                            self.app.cell_height//2)//self.app.cell_height+1
        if self.on_coin():
```

```
            self.eat_coin()

    def draw(self):
        pygame.draw.circle(self.app.screen, PLAYER_COLOUR, (int(self.pix_pos.x),
                                                            int(self.pix_pos.y)),
    self.app.cell_width//2-2)

        # Drawing player lives
        for x in range(self.lives):
            pygame.draw.circle(self.app.screen, PLAYER_COLOUR, (30 + 20*x, HEIGHT -
    15), 7)

        # Drawing the grid pos rect
        # pygame.draw.rect(self.app.screen, RED, (self.grid_pos[0]*self.app.
    cell_width+TOP_BOTTOM_BUFFER//2,
        #                                         self.grid_pos[1]*self.app.
    cell_height+TOP_BOTTOM_BUFFER//2, self.app.cell_width, self.app.cell_height), 1)

    def on_coin(self):
        if self.grid_pos in self.app.coins:
            if int(self.pix_pos.x+TOP_BOTTOM_BUFFER//2) % self.app.cell_width == 0:
                if self.direction == vec(1, 0) or self.direction == vec(-1, 0):
                    return True
            if int(self.pix_pos.y+TOP_BOTTOM_BUFFER//2) % self.app.cell_height == 0:
                if self.direction == vec(0, 1) or self.direction == vec(0, -1):
                    return True
        return False

    def eat_coin(self):
        self.app.coins.remove(self.grid_pos)
        self.current_score += 1

    def move(self, direction):
        self.stored_direction = direction

    def get_pix_pos(self):
        return vec((self.grid_pos[0]*self.app.cell_width)+TOP_BOTTOM_BUFFER//2+self.
    app.cell_width//2,
                   (self.grid_pos[1]*self.app.cell_height) +
                   TOP_BOTTOM_BUFFER//2+self.app.cell_height//2)

        print(self.grid_pos, self.pix_pos)

    def time_to_move(self):
        if int(self.pix_pos.x+TOP_BOTTOM_BUFFER//2) % self.app.cell_width == 0:
            if self.direction == vec(1, 0) or self.direction == vec(-1, 0) or self.
    direction == vec(0, 0):
                return True
        if int(self.pix_pos.y+TOP_BOTTOM_BUFFER//2) % self.app.cell_height == 0:
            if self.direction == vec(0, 1) or self.direction == vec(0, -1) or self.
    direction == vec(0, 0):
                return True

    def can_move(self):
        for wall in self.app.walls:
            if vec(self.grid_pos+self.direction) == wall:
                return False
        return True
```

A.4 File "enemy_class.py"

```
import pygame
import random
```

```
3  from settings import *
4
5  vec = pygame.math.Vector2
6
7
8  class Enemy:
9      def __init__(self, app, pos, number):
10         self.app = app
11         self.grid_pos = pos
12         self.starting_pos = [pos.x, pos.y]
13         self.pix_pos = self.get_pix_pos()
14         self.radius = int(self.app.cell_width//2.3)
15         self.number = number
16         self.colour = self.set_colour()
17         self.direction = vec(0, 0)
18         self.personality = self.set_personality()
19         self.target = None
20         self.speed = self.set_speed()
21
22     def update(self):
23         self.target = self.set_target()
24         if self.target != self.grid_pos:
25             self.pix_pos += self.direction * self.speed
26             if self.time_to_move():
27                 self.move()
28
29         # Setting grid position in reference to pix position
30         self.grid_pos[0] = (self.pix_pos[0]-TOP_BOTTOM_BUFFER +
31                             self.app.cell_width//2)//self.app.cell_width+1
32         self.grid_pos[1] = (self.pix_pos[1]-TOP_BOTTOM_BUFFER +
33                             self.app.cell_height//2)//self.app.cell_height+1
34
35     def draw(self):
36         pygame.draw.circle(self.app.screen, self.colour,
37                            (int(self.pix_pos.x), int(self.pix_pos.y)), self.radius)
38
39     def set_speed(self):
40         if self.personality in ["speedy", "scared"]:
41             speed = 2
42         else:
43             speed = 1
44         return speed
45
46     def set_target(self):
47         if self.personality == "speedy" or self.personality == "slow":
48             return self.app.player.grid_pos
49         else:
50             if self.app.player.grid_pos[0] > COLS//2 and self.app.player.grid_pos[1]
   > ROWS//2:
51                 return vec(1, 1)
52             if self.app.player.grid_pos[0] > COLS//2 and self.app.player.grid_pos[1]
   < ROWS//2:
53                 return vec(1, ROWS-2)
54             if self.app.player.grid_pos[0] < COLS//2 and self.app.player.grid_pos[1]
   > ROWS//2:
55                 return vec(COLS-2, 1)
56             else:
57                 return vec(COLS-2, ROWS-2)
58
59     def time_to_move(self):
60         if int(self.pix_pos.x+TOP_BOTTOM_BUFFER//2) % self.app.cell_width == 0:
61             if self.direction == vec(1, 0) or self.direction == vec(-1, 0) or self.
   direction == vec(0, 0):
62                 return True
63         if int(self.pix_pos.y+TOP_BOTTOM_BUFFER//2) % self.app.cell_height == 0:
```

```
64              if self.direction == vec(0, 1) or self.direction == vec(0, -1) or self.
    direction == vec(0, 0):
65                  return True
66          return False
67
68      def move(self):
69          if self.personality == "random":
70              self.direction = self.get_random_direction()
71          if self.personality == "slow":
72              self.direction = self.get_path_direction(self.target)
73          if self.personality == "speedy":
74              self.direction = self.get_path_direction(self.target)
75          if self.personality == "scared":
76              self.direction = self.get_path_direction(self.target)
77
78      def get_path_direction(self, target):
79          next_cell = self.find_next_cell_in_path(target)
80          xdir = next_cell[0] - self.grid_pos[0]
81          ydir = next_cell[1] - self.grid_pos[1]
82          return vec(xdir, ydir)
83
84      def find_next_cell_in_path(self, target):
85          path = self.BFS([int(self.grid_pos.x), int(self.grid_pos.y)], [
86                           int(target[0]), int(target[1])])
87          return path[1]
88
89      def BFS(self, start, target):
90          grid = [[0 for x in range(28)] for x in range(30)]
91          for cell in self.app.walls:
92              if cell.x < 28 and cell.y < 30:
93                  grid[int(cell.y)][int(cell.x)] = 1
94          queue = [start]
95          path = []
96          visited = []
97          while queue:
98              current = queue[0]
99              queue.remove(queue[0])
100             visited.append(current)
101             if current == target:
102                 break
103             else:
104                 neighbours = [[0, -1], [1, 0], [0, 1], [-1, 0]]
105                 for neighbour in neighbours:
106                     if neighbour[0]+current[0] >= 0 and neighbour[0] + current[0] <
    len(grid[0]):
107                         if neighbour[1]+current[1] >= 0 and neighbour[1] + current[1]
     < len(grid):
108                             next_cell = [neighbour[0] + current[0], neighbour[1] +
    current[1]]
109                             if next_cell not in visited:
110                                 if grid[next_cell[1]][next_cell[0]] != 1:
111                                     queue.append(next_cell)
112                                     path.append({"Current": current, "Next":
    next_cell})
113         shortest = [target]
114         while target != start:
115             for step in path:
116                 if step["Next"] == target:
117                     target = step["Current"]
118                     shortest.insert(0, step["Current"])
119         return shortest
120
121     def get_random_direction(self):
122         while True:
123             number = random.randint(-2, 1)
```

```
124             if number == -2:
125                 x_dir, y_dir = 1, 0
126             elif number == -1:
127                 x_dir, y_dir = 0, 1
128             elif number == 0:
129                 x_dir, y_dir = -1, 0
130             else:
131                 x_dir, y_dir = 0, -1
132             next_pos = vec(self.grid_pos.x + x_dir, self.grid_pos.y + y_dir)
133             if next_pos not in self.app.walls:
134                 break
135         return vec(x_dir, y_dir)
136 
137     def get_pix_pos(self):
138         return vec((self.grid_pos.x*self.app.cell_width)+TOP_BOTTOM_BUFFER//2+self.
    app.cell_width//2,
139                    (self.grid_pos.y*self.app.cell_height)+TOP_BOTTOM_BUFFER//2 +
140                    self.app.cell_height//2)
141 
142     def set_colour(self):
143         if self.number == 0:
144             return (43, 78, 203)
145         if self.number == 1:
146             return (197, 200, 27)
147         if self.number == 2:
148             return (189, 29, 29)
149         if self.number == 3:
150             return (215, 159, 33)
151 
152     def set_personality(self):
153         if self.number == 0:
154             return "speedy"
155         elif self.number == 1:
156             return "slow"
157         elif self.number == 2:
158             return "random"
159         else:
160             return "scared"
```

A.5 File "app_class.py"

```
1  import pygame
2  import sys
3  import copy
4  from settings import *
5  from player_class import *
6  from enemy_class import *
7  
8  
9  pygame.init()
10 vec = pygame.math.Vector2
11 
12 
13 class App:
14     def __init__(self):
15         self.screen = pygame.display.set_mode((WIDTH, HEIGHT))
16         self.clock = pygame.time.Clock()
17         self.running = True
18         self.state = 'start'
19         self.cell_width = MAZE_WIDTH//COLS
20         self.cell_height = MAZE_HEIGHT//ROWS
21         self.walls = []
22         self.coins = []
```

```
23          self.enemies = []
24          self.e_pos = []
25          self.p_pos = None
26          self.load()
27          self.player = Player(self, vec(self.p_pos))
28          self.make_enemies()
29
30      def run(self):
31          while self.running:
32              if self.state == 'start':
33                  self.start_events()
34                  self.start_update()
35                  self.start_draw()
36              elif self.state == 'playing':
37                  self.playing_events()
38                  self.playing_update()
39                  self.playing_draw()
40              elif self.state == 'game over':
41                  self.game_over_events()
42                  self.game_over_update()
43                  self.game_over_draw()
44              else:
45                  self.running = False
46              self.clock.tick(FPS)
47          pygame.quit()
48          sys.exit()
49
50  ############################ HELPER FUNCTIONS ###################################
51
52      def draw_text(self, words, screen, pos, size, colour, font_name, centered=False):
53          font = pygame.font.SysFont(font_name, size)
54          text = font.render(words, False, colour)
55          text_size = text.get_size()
56          if centered:
57              pos[0] = pos[0]-text_size[0]//2
58              pos[1] = pos[1]-text_size[1]//2
59          screen.blit(text, pos)
60
61      def load(self):
62          self.background = pygame.image.load('maze.png')
63          self.background = pygame.transform.scale(self.background, (MAZE_WIDTH,
    MAZE_HEIGHT))
64
65          # Opening walls file
66          # Creating walls list with co-ords of walls
67          # stored as  a vector
68          with open("walls.txt", 'r') as file:
69              for yidx, line in enumerate(file):
70                  for xidx, char in enumerate(line):
71                      if char == "1":
72                          self.walls.append(vec(xidx, yidx))
73                      elif char == "C":
74                          self.coins.append(vec(xidx, yidx))
75                      elif char == "P":
76                          self.p_pos = [xidx, yidx]
77                      elif char in ["2", "3", "4", "5"]:
78                          self.e_pos.append([xidx, yidx])
79                      elif char == "B":
80                          pygame.draw.rect(self.background, BLACK, (xidx*self.
    cell_width, yidx*self.cell_height,
81                                                                    self.cell_width,
    self.cell_height))
82
83      def make_enemies(self):
84          for idx, pos in enumerate(self.e_pos):
```

```
85              self.enemies.append(Enemy(self, vec(pos), idx))
86
87      def draw_grid(self):
88          for x in range(WIDTH//self.cell_width):
89              pygame.draw.line(self.background, GREY, (x*self.cell_width, 0),
90                               (x*self.cell_width, HEIGHT))
91          for x in range(HEIGHT//self.cell_height):
92              pygame.draw.line(self.background, GREY, (0, x*self.cell_height),
93                               (WIDTH, x*self.cell_height))
94          # for coin in self.coins:
95          #     pygame.draw.rect(self.background, (167, 179, 34), (coin.x*self.
    cell_width,
96          #                                                        coin.y*self.
    cell_height, self.cell_width, self.cell_height))
97
98      def reset(self):
99          self.player.lives = 3
100         self.player.current_score = 0
101         self.player.grid_pos = vec(self.player.starting_pos)
102         self.player.pix_pos = self.player.get_pix_pos()
103         self.player.direction *= 0
104         for enemy in self.enemies:
105             enemy.grid_pos = vec(enemy.starting_pos)
106             enemy.pix_pos = enemy.get_pix_pos()
107             enemy.direction *= 0
108
109         self.coins = []
110         with open("walls.txt", 'r') as file:
111             for yidx, line in enumerate(file):
112                 for xidx, char in enumerate(line):
113                     if char == 'C':
114                         self.coins.append(vec(xidx, yidx))
115         self.state = "playing"
116
117
118 ########################### INTRO FUNCTIONS ####################################
119
120     def start_events(self):
121         for event in pygame.event.get():
122             if event.type == pygame.QUIT:
123                 self.running = False
124             if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
125                 self.state = 'playing'
126
127     def start_update(self):
128         pass
129
130     def start_draw(self):
131         self.screen.fill(BLACK)
132         self.draw_text('PUSH SPACE BAR', self.screen, [
133                        WIDTH//2, HEIGHT//2-50], START_TEXT_SIZE, (170, 132, 58),
    START_FONT, centered=True)
134         self.draw_text('1 PLAYER ONLY', self.screen, [
135                        WIDTH//2, HEIGHT//2+50], START_TEXT_SIZE, (44, 167, 198),
    START_FONT, centered=True)
136         self.draw_text('HIGH SCORE', self.screen, [4, 0],
137                        START_TEXT_SIZE, (255, 255, 255), START_FONT)
138         pygame.display.update()
139
140 ########################### PLAYING FUNCTIONS ##################################
141
142     def playing_events(self):
143         for event in pygame.event.get():
144             if event.type == pygame.QUIT:
145                 self.running = False
```

```
146             if event.type == pygame.KEYDOWN:
147                 if event.key == pygame.K_LEFT:
148                     self.player.move(vec(-1, 0))
149                 if event.key == pygame.K_RIGHT:
150                     self.player.move(vec(1, 0))
151                 if event.key == pygame.K_UP:
152                     self.player.move(vec(0, -1))
153                 if event.key == pygame.K_DOWN:
154                     self.player.move(vec(0, 1))
155 
156     def playing_update(self):
157         self.player.update()
158         for enemy in self.enemies:
159             enemy.update()
160 
161         for enemy in self.enemies:
162             if enemy.grid_pos == self.player.grid_pos:
163                 self.remove_life()
164 
165     def playing_draw(self):
166         self.screen.fill(BLACK)
167         self.screen.blit(self.background, (TOP_BOTTOM_BUFFER//2, TOP_BOTTOM_BUFFER
    //2))
168         self.draw_coins()
169         # self.draw_grid()
170         self.draw_text('CURRENT SCORE: {}'.format(self.player.current_score),
171                        self.screen, [60, 0], 18, WHITE, START_FONT)
172         self.draw_text('HIGH SCORE: 0', self.screen, [WIDTH//2+60, 0], 18, WHITE,
    START_FONT)
173         self.player.draw()
174         for enemy in self.enemies:
175             enemy.draw()
176         pygame.display.update()
177 
178     def remove_life(self):
179         self.player.lives -= 1
180         if self.player.lives == 0:
181             self.state = "game over"
182         else:
183             self.player.grid_pos = vec(self.player.starting_pos)
184             self.player.pix_pos = self.player.get_pix_pos()
185             self.player.direction *= 0
186             for enemy in self.enemies:
187                 enemy.grid_pos = vec(enemy.starting_pos)
188                 enemy.pix_pos = enemy.get_pix_pos()
189                 enemy.direction *= 0
190 
191     def draw_coins(self):
192         for coin in self.coins:
193             pygame.draw.circle(self.screen, (124, 123, 7),
194                                (int(coin.x*self.cell_width)+self.cell_width//2+
    TOP_BOTTOM_BUFFER//2,
195                                 int(coin.y*self.cell_height)+self.cell_height//2+
    TOP_BOTTOM_BUFFER//2), 5)
196 
197 ############################ GAME OVER FUNCTIONS ################################
198 
199     def game_over_events(self):
200         for event in pygame.event.get():
201             if event.type == pygame.QUIT:
202                 self.running = False
203             if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
204                 self.reset()
205             if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE:
206                 self.running = False
```

```
207
208     def game_over_update(self):
209         pass
210
211     def game_over_draw(self):
212         self.screen.fill(BLACK)
213         quit_text = "Press the escape button to QUIT"
214         again_text = "Press SPACE bar to PLAY AGAIN"
215         self.draw_text("GAME OVER", self.screen, [WIDTH//2, 100],  52, RED, "arial",
    centered=True)
216         self.draw_text(again_text, self.screen, [
217                         WIDTH//2, HEIGHT//2],  36, (190, 190, 190), "arial", centered=
    True)
218         self.draw_text(quit_text, self.screen, [
219                         WIDTH//2, HEIGHT//1.5],  36, (190, 190, 190), "arial",
    centered=True)
220         pygame.display.update()
```

A.6 File "main.py"

```
1 from app_class import *
2
3 if __name__ == '__main__':
4     app = App()
5     app.run()
```

Appendice B

Codice della revisione dell'implementazione di base

Di seguito sono mostrati i codici sorgente della revisione dell'implementazione di base.

B.1 File "walls.txt"

```
1 11111111111111111111111111111
2 1DDDDDXDDDDDD11DDDDDDXDDDDD1
3 1D1111D11111D11D11111D1111D1
4 1P1111D11111D11D11111D1111P1
5 1D1111D11111D11D11111D1111D1
6 1XDDDDXDDXDDXDDXDDXDDXDDDDX1
7 1D1111D11D11111111D11D1111D1
8 1D1111D11D11111111D11D1111D1
9 1DDDDDX11DDDD11DDDD11XDDDDD1
10 111111D11111011011111D111111
11 111111D11111011011111D111111
12 111111D11000Y00Y00011D100000
13 111111D110111BB111011D111111
14 111111D11015000041011D111111
15 000000RDDY10000001YDDL000000
16 111111D11012000031011D111111
17 111111D11011111111011D111111
18 111111D11Y00000000Y11D111111
19 111111D11011111111011D111111
20 111111D11011111111011D111111
21 1DDDDDXDDXDDD11DDDXDDXDDDDD1
22 1D1111D11111D11D11111D1111D1
23 1D1111D11111D11D11111D1111D1
24 1PDD11XDDXDDX00XDDXDDX11DDP1
25 111D11D11D11111111D11D11D111
26 111D11D11D11111111D11D11D111
27 1DDXDDD11DDDD11DDDD11DDDXDD1
28 1D111111111D11D111111111D1
29 1D111111111D11D111111111D1
30 1DDDDDDDDDDDXDDXDDDDDDDDDD1
31 11111111111111111111111111111
```

B.2 File "settings.py"

```
from pygame.math import Vector2 as vec

# Screen settings
WIDTH, HEIGHT = 610, 670
FPS = 60
TOP_BOTTOM_BUFFER = 50
MAZE_WIDTH, MAZE_HEIGHT = WIDTH-TOP_BOTTOM_BUFFER, HEIGHT-TOP_BOTTOM_BUFFER

# Colour settings
BLACK = (0, 0, 0)
RED = (255, 0, 0)
WHITE = (255, 255, 255)
GREY = (60, 60, 60)
BLUE = (0, 0, 255)

CLYDE_C = (255, 0, 0)
PINKY_C = (255, 184, 255)
INKY_C = (0, 255, 255)
BLINKY_C = (255, 184, 82)

PLAYER_COLOUR = (255, 255, 0)
DOT_PELLET_COLOUR = (255, 185, 175)

VICTORY_COLOUR = (204, 164, 61)

# Font settings
TITLE_TEXT_SIZE = 48

START_TEXT_SIZE = 16
START_FONT = 'arial black'

# Player settings
PLAYER_START_POS = vec(13.5, 23)
PLAYER_SPEED = 1.1

# Enemies settings
CLYDE_START_POS = vec(13.5, 11)
PINKY_START_POS = vec(13.5, 14)
INKY_START_POS = vec(12, 14)
BLINKY_START_POS = vec(15, 14)
ENEMIES_SPEED = 0.8

# Points settings
DOT_PTS = 10
PELLET_PTS = 50
VULNERABLE_GHOST_PTS = 200
CHERRY_PTS = 100
STRAWBERRY_PTS = 300
ORANGE_PTS = 500
APPLE_PTS = 700
MELON_PTS = 1000
GALAXIAN_BOSS_PTS = 2000
BELL_PTS = 3000
KEY_PTS = 5000

# Help settings
GRID = True
PLAYER_POS_CELL = True
ENEMY_POS_CELL = True
```

B.3 File "player_class.py"

```
import pygame
import time
import copy
from settings import *

vec = pygame.math.Vector2

class Player:
    # Costruttore
    def __init__(self, app):
        self.app = app
        self.grid_pos = copy.deepcopy(PLAYER_START_POS)
        self.starting_pos = copy.deepcopy(PLAYER_START_POS)
        self.pix_pos = self.get_pix_pos()
        self.direction = vec(1, 0)
        self.stored_direction = None
        self.able_to_move = True
        self.current_score = 0
        self.eaten_dots = 0
        self.speed = 1
        self.lives = 3
        self.counter = 1

    # Update
    def update(self):
        # Se il player può muoversi, allora effettua un moviemnto
        if self.able_to_move:
            self.pix_pos += self.direction*self.speed
        # Se si trova al  centro della casella, e pertanto può cambiare  la sua direzione
        if self.time_to_move():
            # Cambia la direzione
            if self.stored_direction != None:
                self.direction = self.stored_direction
            # Controlla se può continuare a muoversi
            self.able_to_move = self.can_move()
        # Controlla se ha preso uno dei due corridoi che porta dall'altra parte della mappa
        if self.grid_pos == vec(28, 14) and self.stored_direction == vec(1, 0):
            self.grid_pos = vec(0, 14)
            self.pix_pos[0] = (self.grid_pos[0]-1)*self.app.cell_width+TOP_BOTTOM_BUFFER
        if self.grid_pos == vec(-1, 14) and self.stored_direction == vec(-1, 0):
            self.grid_pos = vec(27, 14)
            self.pix_pos[0] = (self.grid_pos[0]-1)*self.app.cell_width+TOP_BOTTOM_BUFFER
        # Calcola la posizione del fantasma nella griglia
        self.grid_pos[0] = (self.pix_pos[0]-TOP_BOTTOM_BUFFER +
                            self.app.cell_width//2)//self.app.cell_width+1
        self.grid_pos[1] = (self.pix_pos[1]-TOP_BOTTOM_BUFFER +
                            self.app.cell_height//2)//self.app.cell_height+1
        # Se si trova su un dot, allora chiama la funzione per mangiare il dot
        if self.on_dot():
            self.eat_dot()
        # Se si trova su un pellet, allora chiama la funzione per mangiare un pellet
        if self.on_pellet():
            self.eat_pellet()

    # Funzione che disegna il player sullo schermo
    # La seconda draw disegna le vite di cui dispone il player
    def draw(self):
```

```
58          pygame.draw.circle(self.app.screen, PLAYER_COLOUR, (int(self.pix_pos.x), int(
    self.pix_pos.y)), self.app.cell_width//2-2)
59          for x in range(self.lives):
60              pygame.draw.circle(self.app.screen, PLAYER_COLOUR, (30 + 20*x, HEIGHT -
    15), 7)
61
62      # Verifica se il player si trova si un dot
63      # (ossia se si trova al centro della casella dove si trova il dot)
64      def on_dot(self):
65          if self.grid_pos in self.app.dots:
66              if int(self.pix_pos.x+TOP_BOTTOM_BUFFER//2) % self.app.cell_width == 0:
67                  if self.direction == vec(1, 0) or self.direction == vec(-1, 0):
68                      return True
69              if int(self.pix_pos.y+TOP_BOTTOM_BUFFER//2) % self.app.cell_height == 0:
70                  if self.direction == vec(0, 1) or self.direction == vec(0, -1):
71                      return True
72          return False
73
74      # Verifica se il player si trova si un pellet
75      # (ossia se si trova al centro della casella dove si trova il pellet)
76      def on_pellet(self):
77          if self.grid_pos in self.app.pellets:
78              if int(self.pix_pos.x+TOP_BOTTOM_BUFFER//2) % self.app.cell_width == 0:
79                  if self.direction == vec(1, 0) or self.direction == vec(-1, 0):
80                      return True
81              if int(self.pix_pos.y+TOP_BOTTOM_BUFFER//2) % self.app.cell_height == 0:
82                  if self.direction == vec(0, 1) or self.direction == vec(0, -1):
83                      return True
84          return False
85
86      # Mangia il dot su cui si trova
87      # (rimuove il pellet dall'array e incrementa il numero di dot mangiati)
88      def eat_dot(self):
89          self.app.dots.remove(self.grid_pos)
90          self.current_score += DOT_PTS
91          self.eaten_dots += 1
92
93      # Mangia il pellet su cui si trova (rimuove il pellet dall'array e incrementa il
    numero di dot mangiati)
94      # Inoltre setta tutti i fantasmi sullo stato di "Frightened"
95      def eat_pellet(self):
96          initial_time = time.clock()
97          self.app.pellets.remove(self.grid_pos)
98          self.current_score += PELLET_PTS
99          self.eaten_dots += 1
100         # Per ogni fantasma vado a settare il suo stato in "Frightened", a cambiargli
     colore e a decrementare la sua velocita'
101         for enemy in self.app.enemies:
102             # Controllo se il fantasma è nello stato di "Chase"
103             if enemy.state == "Chase":
104                 enemy.state = "Frightened"
105                 # Se si trova all'esterno della zona di spawn, allora inverto la sua
    direzione
106                 if enemy.outside:
107                     enemy.direction *= -1
108                 enemy.modifier = 0.75
109                 # Alloco alcune variabili all'interno dell'oggetto del fantasma
110                 enemy.colour = BLUE
111                 enemy.initial_time = initial_time
112                 enemy.counter = 0
113
114     # Mangia il fantasma concui si incrocia, e setta il fantasma nello stato di "
    Eaten"
115     # Inoltre incrementa il numero di fantasmi mangiati e calcola il percorso per
    arrivare alla zona di spawn
```

```
116     def eat_enemy(self, enemy):
117         self.current_score += VULNERABLE_GHOST_PTS*self.counter
118         self.counter += 1
119         enemy.state = "Eaten"
120         enemy.colour = enemy.set_colour()
121         enemy.modifier = 1
122 
123     # Salva la direzione appena ricevuta
124     def move(self, direction):
125         self.stored_direction = direction
126 
127 ########## HELPER FUNCTIONS ##########
128 
129     # Calcola la posizione in cui deve disegnare il player
130     def get_pix_pos(self):
131         return vec((self.grid_pos[0]*self.app.cell_width)+TOP_BOTTOM_BUFFER//2+self.
    app.cell_width//2,
132                    (self.grid_pos[1]*self.app.cell_height) +
133                    TOP_BOTTOM_BUFFER//2+self.app.cell_height//2)
134 
135         print(self.grid_pos, self.pix_pos)
136 
137     # Controlla se il player può muoversi
138     # (controlla se si trova al centro della casella della griglia)
139     def time_to_move(self):
140         if int(self.pix_pos.x+TOP_BOTTOM_BUFFER//2) % self.app.cell_width == 0:
141             if self.direction == vec(1, 0) or self.direction == vec(-1, 0) or self.
    direction == vec(0, 0):
142                 return True
143         if int(self.pix_pos.y+TOP_BOTTOM_BUFFER//2) % self.app.cell_height == 0:
144             if self.direction == vec(0, 1) or self.direction == vec(0, -1) or self.
    direction == vec(0, 0):
145                 return True
146 
147     # Controlla se il fantasma può continuare a muoversi in una certa direzione
148     def can_move(self):
149         if vec(self.grid_pos+self.direction) in self.app.walls:
150             return False
151         return True
```

B.4 File "enemy_class.py"

```
1  import math
2  import numpy
3  import pygame
4  import random
5  import time
6  import copy
7  from settings import *
8  from pygame.math import Vector2 as vec
9 
10 class Enemy:
11     # Costruttore
12     def __init__(self, app, name, number, outside):
13         self.app = app
14         self.name = name
15         self.number = number
16         self.able_to_move = True
17         self.grid_pos = self.get_grid_pos()
18         self.starting_pos = self.get_grid_pos()
19         self.pix_pos = self.get_pix_pos()
20         self.colour = self.set_colour()
21         self.direction = vec(1, 0)
```

```
22          self.stored_direction = self.direction
23          self.outside = outside
24          self.state = "Chase"
25          self.modifier = 1
26
27      # Update
28      def update(self):
29          # Se il nemico può muoversi, allora effettua un movimento
30          if self.able_to_move:
31              self.pix_pos += self.direction*ENEMIES_SPEED*self.modifier
32          # Se si trova al centro della casella, e pertanto può cambiare la sua
    direzione
33          if self.time_to_move():
34              # Controlla se si può muovere
35              self.able_to_move = self.can_move()
36              # Se non si può muovere di due caselle avanti oppure si trova vicino ad
    un incrocio
37              # Decide la nuova direzione che terra‘ da parte
38              if (not self.can_move_next()) or self.check_intersection_near():
39                  self.move()
40              # Se non si può muovere di una casella avanti oppure si trova su un
    incrocio
41              # usa la direzione che aveva deciso in precedenza
42              if (not self.able_to_move) or self.check_intersection():
43                  self.direction = self.stored_direction
44              # Se deve ancora uscire dalla zona di spawn
45              # Calcola la direzione e la usa subito
46              if (not self.outside):
47                  self.move()
48                  self.direction = self.stored_direction
49          # Controlla se ha preso uno dei due corridoi che porta dall'altra parte della
     mappa
50          if self.grid_pos == vec(28, 14) and self.direction == vec(1, 0):
51              self.grid_pos = vec(0, 14)
52              self.pix_pos[0] = (self.grid_pos[0]-1)*self.app.cell_width+
    TOP_BOTTOM_BUFFER
53          if self.grid_pos == vec(-1, 14) and self.direction == vec(-1, 0):
54              self.grid_pos = vec(27, 14)
55              self.pix_pos[0] = (self.grid_pos[0]-1)*self.app.cell_width+
    TOP_BOTTOM_BUFFER
56          # Calcola la posizione del fantasma nella griglia
57          self.grid_pos[0] = (self.pix_pos[0]-TOP_BOTTOM_BUFFER)//self.app.cell_width+1
58          self.grid_pos[1] = (self.pix_pos[1]-TOP_BOTTOM_BUFFER)//self.app.cell_height
    +1
59          # Se il fantasma è nello stato di "Frightened", chiama la funzione per
    resettarlo
60          if self.state == "Frightened":
61              self.reset_to_chase()
62          # Se il fantasma è stato mangiato ed è tornato della zona di spawn allora
    torna allo stato normale
63          if self.state == "Eaten" and (self.grid_pos == vec(13, 14) or self.grid_pos
    == vec(14, 14)):
64              self.state = "Chase"
65              self.modifier = 1
66              self.outside = False
67
68      # Nel caso in cui il fantasma non sia ancora uscito dalla zona di spawn
69      # Controlla solo se il fantasma è entrato nello stato di "Frightened"
70      def update_state_only(self):
71          if self.state == "Frightened":
72              self.reset_to_chase()
73
74      # Porta i fantasmi dallo stato di "Frightened" allo stato di "Chase"
75      # Si occupa anche di far cambiare colore ai fantasmi
76      def reset_to_chase(self):
```

```
77          # Controlla se sono passati 6 secondi da quando i
78          # Fantasmi sono entrati nello stato di "Frightened"
79          if (time.clock()-self.initial_time) > 6.0:
80              # Effettua ad intervalli di 1 secondo il cambio
81              # Di colore da bianco a blue viceversa
82              if (time.clock()-self.initial_time) > (6.0+(self.counter/2)) and self.
    counter%2 == 0:
83                  self.counter += 1
84                  self.colour = WHITE
85              elif (time.clock()-self.initial_time) > (6.0+(self.counter/2)) and self.
    counter%2 == 1:
86                  self.counter += 1
87                  self.colour = BLUE
88          # Dopo 11 secondi resetta il fantasma allo stato di "Chase"
89          if (time.clock()-self.initial_time) > 9.0:
90              self.state = "Chase"
91              self.modifier = 1
92              self.colour = self.set_colour()
93              self.app.player.counter = 1
94
95      # Chiama la funzione per muoversi in base allo stato o al nome del fantasma
96      def move(self):
97          if self.state == "Frightened" and self.outside:
98              self.frightened()
99          elif self.state == "Eaten":
100             self.choose_direction(vec(13.5, 14))
101         else:
102             if self.name == "Clyde":
103                 target_pos = self.chase_clyde()
104             elif self.name == "Pinky":
105                 target_pos = self.chase_pinky()
106             elif self.name == "Inky":
107                 target_pos = self.chase_inky()
108             elif self.name == "Blinky":
109                 target_pos = self.chase_blinky()
110             self.choose_direction(target_pos)
111
112     # Funzione che disegna il fantasma sullo schermo
113     # La seconda draw viene usata quando un fantasma viene mangiato
114     def draw(self):
115         if self.state == "Eaten":
116             pygame.draw.circle(self.app.screen, self.colour,
117                                (int(self.pix_pos.x), int(self.pix_pos.y)), self.app.
    cell_width//2-2, 1)
118         else:
119             pygame.draw.circle(self.app.screen, self.colour,
120                                (int(self.pix_pos.x), int(self.pix_pos.y)), self.app.
    cell_width//2-2)
121
122
123     '''
124     Nel caso in cui due o più direzioni danno stesso valore delta,
125     l'ordine di scleta di quale direzione prendere è
126     Sopra -> Sinistra -> Sotto -> Destra
127     '''
128
129     # Quando sono spaventati, prenderanno una direzione diversa da quella da cui
    vengono
130     # Per decidere quale direzione prende, si sceglie una direzione inziale
131     # Nel caso in cui non vada bene, si ruota in senso antiorario
132     def frightened(self):
133         directions = [vec(0, -1), vec(-1, 0), vec(0, 1), vec(1, 0)]
134         starting_idx = random.randint(0, 3)
135         actual_pos = self.grid_pos+self.direction
136         for i in range(4):
```

```
137             direction = directions[(starting_idx + i) % 4]
138             if self.can_move_certain_direction(actual_pos, direction) and actual_pos+
    direction != self.grid_pos:
139                 self.stored_direction = direction
140 
141     # Decide quale direzione deve prendere il fantasma usando la posizione target che
     deve raggiungere
142     # Esclude la direzione da cui il fantasma proviene
143     def choose_direction(self, target_pos):
144         distances = []
145         directions = [vec(0, -1), vec(-1, 0), vec(0, 1), vec(1, 0)]
146         # Controllo se il fantasma è al di fuori della zona di spawn oppure no
147         # Se si' considero come posizione effettuiva la sua posizione più la sua
    direzione
148         # Altrimenti considero la sua posizione effettiva
149         if (not self.outside) or (not self.can_move()):
150             actual_pos = self.grid_pos
151         else:
152             actual_pos = self.grid_pos+self.direction
153         for direction in directions:
154             # Controllo se si può muovere in una direzione, e se si' calcolo la
    distanza dalla casella target
155             if self.can_move_certain_direction(actual_pos, direction) and self.
    direction != (-1*direction):
156                 # Se il fantasma si trova nella casella (14, 6), considera la
    posizione (14, 33) per
157                 # Calcolare la distanza effettiva dalla casella target
158                 if (actual_pos == vec(14, 6)) and direction == self.app.crossroad_L:
159                     next_pos = vec(14, 33)
160                 # Se il fantasma si trova nella casella (14, 21), considera la
    posizione (14, -6) per
161                 # Calcolare la distanza effettiva dalla casella target
162                 elif (actual_pos == vec(14, 21)) and direction == self.app.
    crossroad_R:
163                     next_pos = vec(14, -6)
164                 else:
165                     next_pos = actual_pos+direction
166                 distances.append(round((next_pos.x-target_pos.x)**2+((next_pos.y-
    target_pos.y))**2))
167             else:
168                 distances.append(3000)
169         # Cerco l'indice dell'elemento con distanza minore, e lo uso per deicdere(0:)
170         m = numpy.argmin(distances)
171         if m == 0:
172             self.stored_direction = vec(0, -1)
173         elif m == 1:
174             self.stored_direction = vec(-1, 0)
175         elif m == 2:
176             self.stored_direction = vec(0, 1)
177         elif m == 3:
178             self.stored_direction = vec(1, 0)
179 
180     # Clyde cerca di raggiungere la posizione in cui si trova il player
181     def chase_clyde(self):
182         if self.grid_pos == (13, 11) or self.grid_pos == (14, 11):
183             self.outside = True
184         if not self.outside:
185             target_pos = vec(13, 11)
186         else:
187             target_pos = self.app.player.grid_pos
188         return target_pos
189 
190     # Pinky punta 4 caselle in avanti rispetto a dove è il player
191     # Ne considera la direzione, e se il player guarda su, il fantasma guardera' 4
    caselle su e 4 a sinistra
```

```
192     # (Behaviur dovuto ad un bug nelle prive versioni arcade)
193     def chase_pinky(self):
194         if self.grid_pos == (13, 11) or self.grid_pos == (14, 11):
195             self.outside = True
196         if not self.outside:
197             target_pos = vec(13, 11)
198         else:
199             target_pos = self.app.player.grid_pos+(self.app.player.direction*4)
200             if self.app.player.direction == vec(0, -1):
201                 target_pos += vec(-4, 0)
202         return target_pos
203 
204     # Inky consuidera una casella avanti al player, punta una freccia da quella
        casella verso Clyde, e poi la ruota di 180 gradi
205     # Ne considera la direzione, e se il player guarda su, il fantasma guardera‘ 4
        caselle su e 4 a sinistra
206     # (Behaviur dovuto ad un bug nelle prive versioni arcade)
207     def chase_inky(self):
208         if self.grid_pos == (13, 11) or self.grid_pos == (14, 11):
209             self.outside = True
210         if not self.outside:
211             target_pos = vec(13, 11)
212         else:
213             cell_pos = self.app.player.grid_pos+self.app.player.direction*2
214             if self.app.player.direction == vec(0, -1):
215                 cell_pos += vec(-2, 0)
216             offset = cell_pos-self.app.enemies[0].grid_pos
217             target_pos = self.app.enemies[0].grid_pos+(2*offset)
218         return target_pos
219 
220     # Blinky si comporta come Clyde, ma se è troppo vicino al plaver (8 blocchi di
        raggio), punta alla casella (30, 0)
221     def chase_blinky(self):
222         if self.grid_pos == (13, 11) or self.grid_pos == (14, 11):
223             self.outside = True
224         if not self.outside:
225             target_pos = vec(13, 11)
226         else:
227             if (self.grid_pos.x-self.app.player.grid_pos.x)**2+(self.grid_pos.x-self.
    app.player.grid_pos.y)**2 > 64:
228                 target_pos = self.app.player.grid_pos
229             else:
230                 target_pos = vec(30, -1)
231         return target_pos
232 
233 ########## HELPER FUNCTIONS ##########
234 
235     # Recupera dal file "setting.py" le coordinate da cui i fantasmi devono partire
236     def get_grid_pos(self):
237         if self.number == 0:
238             return copy.deepcopy(CLYDE_START_POS)
239         elif self.number == 1:
240             return copy.deepcopy(PINKY_START_POS)
241         elif self.number == 2:
242             return copy.deepcopy(INKY_START_POS)
243         elif self.number == 3:
244             return copy.deepcopy(BLINKY_START_POS)
245 
246     # Calcola la posizione in cui deve disegnare il fantasma
247     def get_pix_pos(self):
248         draw_pos_x = self.grid_pos.x*self.app.cell_width+TOP_BOTTOM_BUFFER//2+self.
    app.cell_width//2
249         draw_pos_y = self.grid_pos.y*self.app.cell_height+TOP_BOTTOM_BUFFER//2+self.
    app.cell_height//2
250         return vec(draw_pos_x, draw_pos_y)
```

```
251
252     # Seleziona dal file "settings.py" il colore del fantasma
253     def set_colour(self):
254         if self.number == 0:
255             return CLYDE_C
256         elif self.number == 1:
257             return PINKY_C
258         elif self.number == 2:
259             return INKY_C
260         elif self.number == 3:
261             return BLINKY_C
262
263     # Controlla se il fantasma può muoversi
264     # (controlla se si trova al centro della casella della griglia)
265     def time_to_move(self):
266         if int(self.pix_pos.x+TOP_BOTTOM_BUFFER//2) % self.app.cell_width == 0:
267             if self.direction == vec(1, 0) or self.direction == vec(-1, 0):
268                 return True
269         if int(self.pix_pos.y+TOP_BOTTOM_BUFFER//2) % self.app.cell_height == 0:
270             if self.direction == vec(0, 1) or self.direction == vec(0, -1):
271                 return True
272
273     # Funzione che trasforma in stringa l'array delle distanze delta
274     def distances_array_print(self, distances):
275         n = len(distances)
276         string = "{"
277         for key, distance in enumerate(distances):
278             if key == 0:
279                 string += str(key)+"(Up):"+str(distance)
280             if key == 1:
281                 string += str(key)+"(Left):"+str(distance)
282             if key == 2:
283                 string += str(key)+"(Down):"+str(distance)
284             if key == 3:
285                 string += str(key)+"(Right):"+str(distance)
286             if key+1 != n:
287                 string += "; "
288         return string
289
290 ########## CONDITION CHECK FUNCTIONS ##########
291
292     # Controlla se il fantasma si può muovere in una certa direzione
293     def can_move_certain_direction(self, position, direction):
294         if vec(position+direction) in self.app.walls:
295             if vec(position+direction) in self.app.barrier and ((not self.outside) or
     self.state == "Eaten"):
296                 return True
297             else:
298                 return False
299         return True
300
301     # Controllase il fantasma può continuare a muoversi di due passi in una certa
    direzione
302     def can_move_next(self):
303         if vec(self.grid_pos+(self.direction*2)) in self.app.walls:
304             if vec(self.grid_pos+(self.direction*2)) in self.app.barrier and ((not
    self.outside) or self.state == "Eaten"):
305                 return True
306             else:
307                 return False
308         return True
309
310     # Controlla se il fantasma può continuare a muoversi di un passo in una certa
    direzione
311     def can_move(self):
```

```
312         if vec(self.grid_pos+self.direction) in self.app.walls:
313             if vec(self.grid_pos+self.direction) in self.app.barrier and ((not self.
    outside) or self.state == "Eaten"):
314                 return True
315             else:
316                 return False
317         return True
318 
319     # Controllo se il fantasma si trova su un'intersezione
320     def check_intersection(self):
321         if self.grid_pos in self.app.crossroads:
322             return True
323         if self.grid_pos == self.app.crossroad_L or self.grid_pos == self.app.
    crossroad_R:
324             return True
325         if (self.grid_pos == vec(13, 11) or self.grid_pos == vec(14, 11)) and self.
    state == "Eaten":
326             return True
327         return False
328 
329     # Controllo se il fantasma si trova vicino ad un'intersezione
330     def check_intersection_near(self):
331         if self.grid_pos+self.direction in self.app.crossroads:
332             return True
333         if self.grid_pos+self.direction == self.app.crossroad_L or self.grid_pos+self
    .direction == self.app.crossroad_R:
334             return True
335         if (self.grid_pos+self.direction == vec(13, 11) or self.grid_pos+self.
    direction == vec(14, 11)) and self.state == "Eaten":
336             return True
337         return False
```

B.5 File "app_class.py"

```
1  import pygame
2  import sys
3  import copy
4  import time
5  from settings import *
6  from player_class import *
7  from enemy_class import *
8 
9  vec = pygame.math.Vector2
10 
11 class App:
12     # Costruttore
13     def __init__(self):
14         pygame.init()
15         self.screen = pygame.display.set_mode((WIDTH, HEIGHT))
16         self.clock = pygame.time.Clock()
17         self.running = True
18         self.state = 'start'
19         self.cell_width = MAZE_WIDTH//28
20         self.cell_height = MAZE_HEIGHT//31
21         self.walls = []
22         self.barrier = []
23         self.dots = []
24         self.pellets = []
25         self.crossroads = []
26         self.crossroad_L = None
27         self.crossroad_R = None
28         self.total_dots = 0
29         self.enemies = []
```

```
30          self.enemies_names = ["Clyde", "Pinky", "Inky", "Blinky"]
31          self.load()
32          self.player = Player(self)
33          self.make_enemies()
34
35      # Funzione run: definisce cosa deve fare il gioco in base allo stato in cui si
        trova
36      def run(self):
37          while self.running:
38              self.clock.tick(FPS)
39              if self.state == "start":
40                  self.start_events()
41                  self.start_update()
42                  self.start_draw()
43              elif self.state == "playing":
44                  self.playing_events()
45                  self.playing_update()
46                  self.playing_draw()
47              elif self.state == "game over":
48                  self.game_over_events()
49                  self.game_over_update()
50                  self.game_over_draw()
51              elif self.state == "victory":
52                  self.victory_events()
53                  self.victory_update()
54                  self.victory_draw()
55              else:
56                  self.running = False
57          pygame.quit()
58          sys.exit()
59
60  ########## HELPER FUNCTIONS ##########
61
62      # Disegna il testo sullo schermo nella posizione indicata
63      # Viene usato il colore e il font passati come parametri
64      def draw_text(self, words, screen, pos, size, colour, font_name, centered=False):
65          font = pygame.font.SysFont(font_name, size)
66          text = font.render(words, False, colour)
67          text_size = text.get_size()
68          if centered:
69              pos[0] = pos[0]-text_size[0]//2
70              pos[1] = pos[1]-text_size[1]//2
71          screen.blit(text, pos)
72
73      # Carica l'immagine "maze.png" e legge il file "walls.txt"
74      # Traduce il file "walls.txt" in informazioni riguardo il labirinto
75      def load(self):
76          self.background = pygame.image.load("maze.png")
77          self.background = pygame.transform.scale(self.background, (MAZE_WIDTH,
        MAZE_HEIGHT))
78          # Identifica i punti importanti della mappa
79          # (pareti, barriera, incroci, dots, pellets)
80          with open("walls.txt", "r") as file:
81              for yidx, line in enumerate(file):
82                  for xidx, char in enumerate(line):
83                      if char in ["1", "B"]:
84                          self.walls.append(vec(xidx, yidx))
85                          if char == "B":
86                              self.barrier.append(vec(xidx, yidx))
87                      elif char in ["D", "X", "L", "R", "P"]:
88                          self.total_dots += 1
89                          if char == "P":
90                              self.pellets.append(vec(xidx, yidx))
91                          else:
92                              self.dots.append(vec(xidx, yidx))
```

```
93                              if char == "X":
94                                  self.crossroads.append(vec(xidx, yidx))
95                              elif char == "L":
96                                  self.crossroad_L = vec(xidx, yidx)
97                              elif char == "R":
98                                  self.crossroad_R = vec(xidx, yidx)
99                      elif char == "Y":
100                         self.crossroads.append(vec(xidx, yidx))
101
102     # Crea gli oggetti "enemies" (i fantasmi)
103     def make_enemies(self):
104         for ind, name in enumerate(self.enemies_names):
105             # Se il fantasma si chiama "Clyde" allora si trova all'esterno della
106             # Zona di spawn, altrimenti si trova all'intero della zona di spawn
107             if name == "Clyde":
108                 self.enemies.append(Enemy(self, name, ind, True))
109             else:
110                 self.enemies.append(Enemy(self, name, ind, False))
111
112     # Resetta il gioco
113     def reset(self):
114         self.player = None
115         self.enemies = []
116         self.player = Player(self)
117         self.make_enemies()
118         self.load()
119         self.state = "playing"
120
121 ########## INTRO FUNCTIONS ##########
122
123     def start_events(self):
124         for event in pygame.event.get():
125             if event.type == pygame.QUIT:
126                 self.running = False
127             if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
128                 self.state = "playing"
129
130     def start_update(self):
131         pass
132
133     def start_draw(self):
134         self.screen.fill(BLACK)
135         self.draw_text("PUSH SPACE BAR", self.screen, [
136                         WIDTH//2, HEIGHT//2-50], START_TEXT_SIZE, (170, 132, 58),
    START_FONT, centered=True)
137         self.draw_text("1 PLAYER ONLY", self.screen, [
138                         WIDTH//2, HEIGHT//2+50], START_TEXT_SIZE, (44, 167, 198),
    START_FONT, centered=True)
139         self.draw_text("HIGH SCORE", self.screen, [4, 0],
140                         START_TEXT_SIZE, (255, 255, 255), START_FONT)
141         pygame.display.update()
142
143 ########## PLAYING FUNCTIONS ##########
144
145     def playing_events(self):
146         for event in pygame.event.get():
147             if event.type == pygame.QUIT:
148                 self.running = False
149             if event.type == pygame.KEYDOWN:
150                 if event.key == pygame.K_LEFT:
151                     self.player.move(vec(-1, 0))
152                 if event.key == pygame.K_RIGHT:
153                     self.player.move(vec(1, 0))
154                 if event.key == pygame.K_UP:
155                     self.player.move(vec(0, -1))
```

```
156                 if event.key == pygame.K_DOWN:
157                     self.player.move(vec(0, 1))
158 
159     def playing_update(self):
160         # Effettua l'udpate del player
161         self.player.update()
162         # Effettua lúpdate dei fantasmi
163         for enemy in self.enemies:
164             # Se il fantasma si chiama Inky e sono stati mangiati meno di 30 dots
165             # Allora il fantasma non può essere aggiornato
166             if enemy.name == "Inky" and self.player.eaten_dots < 30:
167                 enemy.update_state_only()
168             # Se il fantasma si chiama Inky e sono stati mangiati meno di 1/3 dei
    dots
169             # Allora il fantasma non può essere aggiornato
170             elif enemy.name == "Blinky" and self.player.eaten_dots < self.total_dots
    //3:
171                 enemy.update_state_only()
172             else:
173                 enemy.update()
174             # Se il fantasma ènella stessa casella del player
175             if self.player.grid_pos == enemy.grid_pos:
176                 # Se il fantasma è nello stato di "Chase"
177                 # Allora viene tolta una vita al player
178                 if enemy.state == "Chase":
179                     self.remove_life()
180                 # Se il fantasma è nello stato di "Frightened"
181                 # Allora il fantasma viene mangiato
182                 elif enemy.state == "Frightened":
183                     self.player.eat_enemy(enemy)
184         # Se il player ha mangiato tutti i dots, allora il player ha vinto
185         if self.player.eaten_dots == self.total_dots:
186             self.state = "victory"
187 
188     def playing_draw(self):
189         self.screen.fill(BLACK)
190         self.screen.blit(self.background, (TOP_BOTTOM_BUFFER//2, TOP_BOTTOM_BUFFER
    //2))
191         self.draw_dots()
192         self.draw_pellets()
193         self.draw_text("CURRENT SCORE: {}".format(self.player.current_score),
194                        self.screen, [60, 0], 18, WHITE, START_FONT)
195         self.player.draw()
196         for enemy in self.enemies:
197             enemy.draw()
198         pygame.display.update()
199 
200     # Rimuove una vita al player e riposiziona player e fantasmi
201     # Se le vite del player vanno a 0, allora il giocatore perdfe
202     def remove_life(self):
203         self.player.lives -= 1
204         if self.player.lives == 0:
205             self.state = "game over"
206         else:
207             self.player.grid_pos = copy.deepcopy(self.player.starting_pos)
208             self.player.pix_pos = self.player.get_pix_pos()
209             self.player.direction = vec(1, 0)
210             for enemy in self.enemies:
211                 enemy.grid_pos = vec(enemy.starting_pos)
212                 enemy.pix_pos = enemy.get_pix_pos()
213                 enemy.direction = vec(1, 0)
214                 if enemy.name == "Clyde":
215                     enemy.outside = True
216                 else:
217                     enemy.outside = False
```

```
218
219     def draw_dots(self):
220         for dot in self.dots:
221             pygame.draw.circle(self.screen, DOT_PELLET_COLOUR,
222                                (int(dot.x*self.cell_width)+self.cell_width//2+
    TOP_BOTTOM_BUFFER//2,
223                                 int(dot.y*self.cell_height)+self.cell_height//2+
    TOP_BOTTOM_BUFFER//2), 2)
224
225     def draw_pellets(self):
226         for pellet in self.pellets:
227             pygame.draw.circle(self.screen, DOT_PELLET_COLOUR,
228                                (int(pellet.x*self.cell_width)+self.cell_width//2+
    TOP_BOTTOM_BUFFER//2,
229                                 int(pellet.y*self.cell_height)+self.cell_height//2+
    TOP_BOTTOM_BUFFER//2), 4)
230
231 ########## GAME OVER FUNCTIONS ##########
232
233     def game_over_events(self):
234         for event in pygame.event.get():
235             if event.type == pygame.QUIT:
236                 self.running = False
237             if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
238                 self.reset()
239             if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE:
240                 self.running = False
241
242     def game_over_update(self):
243         pass
244
245     def game_over_draw(self):
246         self.screen.fill(BLACK)
247         quit_text = "Press the escape button to QUIT"
248         again_text = "Press SPACE bar to PLAY AGAIN"
249         self.draw_text("GAME OVER", self.screen, [WIDTH//2, 100],  52, RED, "arial",
    centered=True)
250         self.draw_text(again_text, self.screen, [
251                        WIDTH//2, HEIGHT//2],  36, (190, 190, 190), "arial", centered=
    True)
252         self.draw_text(quit_text, self.screen, [
253                        WIDTH//2, HEIGHT//1.5],  36, (190, 190, 190), "arial",
    centered=True)
254         pygame.display.update()
255
256 ########## VICTORY FUNCTIONS ##########
257
258     def victory_events(self):
259         for event in pygame.event.get():
260             if event.type == pygame.QUIT:
261                 self.running = False
262             if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
263                 self.reset()
264             if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE:
265                 self.running = False
266
267     def victory_update(self):
268         pass
269
270     def victory_draw(self):
271         self.screen.fill(BLACK)
272         quit_text = "Press the escape button to QUIT"
273         again_text = "Press SPACE bar to PLAY AGAIN"
274         self.draw_text("VICTORY!", self.screen, [WIDTH//2, 100],  52, VICTORY_COLOUR,
     "arial", centered=True)
```

```
275         self.draw_text(again_text, self.screen, [
276                        WIDTH//2, HEIGHT//2],  36, (190, 190, 190), "arial", centered=
    True)
277         self.draw_text(quit_text, self.screen, [
278                        WIDTH//2, HEIGHT//1.5],  36, (190, 190, 190), "arial",
    centered=True)
279         pygame.display.update()
```

B.6 File "main.py"

```
1 from app_class import *
2
3 if __name__ == '__main__':
4     print("*Pac-Man* - Created by Emanuele Izzo")
5     app = App()
6     app.run()
```

Appendice C

Codice dell'implementazione dell'agente programmato

Di seguito sono riportati i codici sorgente dell'implementazione dell'agente programmato. Per evitare ripetizioni, sono mostrati i codici dei soli file che hanno subito modifiche. per i restanti, si fa riferimento ai codici mostrati nell'appendice B.

C.1 File "player_class.py"

```
import pygame
import time
import random
import numpy
from settings import *

vec = pygame.math.Vector2

class Player:
    # Costruttore
    def __init__(self, app):
        self.app = app
        self.grid_pos = PLAYER_START_POS
        self.pix_pos = self.get_pix_pos()
        if random.randint(0, 1) == 1:
            self.direction = vec(1, 0)
        else:
            self.direction = vec(-1, 0)
        self.stored_direction = None
        self.able_to_move = True
        self.current_score = 0
        self.eaten_dots = 0
        self.speed = 1
        self.lives = 1 # 3
        self.counter = 1

    # Update
    def update(self):
        # Se il player può muoversi, allora effettua un moviemnto
        if self.able_to_move:
            self.pix_pos += self.direction*self.speed
```

```
32          # Se si trova al  centro della casella, e pertanto può cambiare  la sua
    direzione
33          if self.time_to_move():
34              # Cambia la direzione
35              if self.stored_direction != None:
36                  self.direction = self.stored_direction
37              # Controlla se può continuare a muoversi
38              self.able_to_move = self.can_move()
39          # Controlla se ha preso uno dei due corridoi che porta dall'altra parte della
     mappa
40          if self.grid_pos == vec(28, 14) and self.stored_direction == vec(1, 0):
41              self.grid_pos = vec(0, 14)
42              self.pix_pos[0] = (self.grid_pos[0]-1)*self.app.cell_width+
    TOP_BOTTOM_BUFFER
43          if self.grid_pos == vec(-1, 14) and self.stored_direction == vec(-1, 0):
44              self.grid_pos = vec(27, 14)
45              self.pix_pos[0] = (self.grid_pos[0]-1)*self.app.cell_width+
    TOP_BOTTOM_BUFFER
46          # Calcola la posizione del fantasma nella griglia
47          self.grid_pos[0] = (self.pix_pos[0]-TOP_BOTTOM_BUFFER +
48                              self.app.cell_width//2)//self.app.cell_width+1
49          self.grid_pos[1] = (self.pix_pos[1]-TOP_BOTTOM_BUFFER +
50                              self.app.cell_height//2)//self.app.cell_height+1
51          # Se si trova su un dot, allora chiama la funzione per mangiare il dot
52          if self.on_dot():
53              self.eat_dot()
54          # Se si trova su un pellet, allora chiama la funzione per mangiare un pellet
55          if self.on_pellet():
56              self.eat_pellet()
57
58      # Funzione che disegna il player sullo schermo
59      # La seconda draw disegna le vite di cui dispone il player
60      def draw(self):
61          pygame.draw.circle(self.app.screen, PLAYER_COLOUR, (int(self.pix_pos.x), int(
    self.pix_pos.y)), self.app.cell_width//2-2)
62          for x in range(self.lives):
63              pygame.draw.circle(self.app.screen, PLAYER_COLOUR, (30 + 20*x, HEIGHT -
    15), 7)
64
65      # Verifica se il player si trova si un dot
66      # (ossia se si trova al centro della casella dove si trova il dot)
67      def on_dot(self):
68          if self.grid_pos in self.app.dots:
69              if int(self.pix_pos.x+TOP_BOTTOM_BUFFER//2) % self.app.cell_width == 0:
70                  if self.direction == vec(1, 0) or self.direction == vec(-1, 0):
71                      return True
72              if int(self.pix_pos.y+TOP_BOTTOM_BUFFER//2) % self.app.cell_height == 0:
73                  if self.direction == vec(0, 1) or self.direction == vec(0, -1):
74                      return True
75          return False
76
77      # Verifica se il player si trova si un pellet
78      # (ossia se si trova al centro della casella dove si trova il pellet)
79      def on_pellet(self):
80          if self.grid_pos in self.app.pellets:
81              if int(self.pix_pos.x+TOP_BOTTOM_BUFFER//2) % self.app.cell_width == 0:
82                  if self.direction == vec(1, 0) or self.direction == vec(-1, 0):
83                      return True
84              if int(self.pix_pos.y+TOP_BOTTOM_BUFFER//2) % self.app.cell_height == 0:
85                  if self.direction == vec(0, 1) or self.direction == vec(0, -1):
86                      return True
87          return False
88
89      # Mangia il dot su cui si trova
90      # (rimuove il pellet dall'array e incrementa il numero di dot mangiati)
```

```
91      def eat_dot(self):
92          self.app.dots.remove(self.grid_pos)
93          self.current_score += DOT_PTS
94          self.eaten_dots += 1
95
96      # Mangia il pellet su cui si trova (rimuove il pellet dall'array e incrementa il
        numero di dot mangiati)
97      # Inoltre setta tutti i fantasmi sullo stato di "Frightened"
98      def eat_pellet(self):
99          initial_time = time.clock()
100         self.app.pellets.remove(self.grid_pos)
101         self.current_score += PELLET_PTS
102         self.eaten_dots += 1
103         # Per ogni fantasma vado a settare il suo stato in "Frightened", a cambiargli
         colore e a decrementare la sua velocita'
104         for enemy in self.app.enemies:
105             # Controllo se il fantasma è nello stato di "Chase"
106             if enemy.state == "Chase":
107                 enemy.state = "Frightened"
108                 # Se si trova all'esterno della zona di spawn, allora inverto la sua
        direzione
109                 if enemy.outside:
110                     enemy.direction *= -1
111                 enemy.modifier = 0.75
112                 # Alloco alcune variabili all'interno dell'oggetto del fantasma
113                 enemy.colour = BLUE
114                 enemy.initial_time = initial_time
115                 enemy.counter = 0
116
117     # Mangia il fantasma concui si incrocia, e setta il fantasma nello stato di "
        Eaten"
118     # Inoltre incrementa il numero di fantasmi mangiati e calcola il percorso per
        arrivare alla zona di spawn
119     def eat_enemy(self, enemy):
120         self.current_score += VULNERABLE_GHOST_PTS*self.counter
121         self.counter += 1
122         enemy.state = "Eaten"
123         enemy.colour = enemy.set_colour()
124         enemy.modifier = 1
125
126     # Salva la direzione appena ricevuta
127     def move(self, direction):
128         self.stored_direction = direction
129
130 ########## NEXT MOVEMENT CALCULATION FUNCTION ##########
131
132     # Calcolo un valore delta per decidere quale direzione il player dovra' prendere
133     # Il valore delta dipende dalla direzione del player e dalla posizione di tutti i
         fantasmi
134     def calculate_direction(self, enemies, dots):
135         # Cerca il fantasma con il nome "Clyde" e lo salva in una variabile
136         for enemy in enemies:
137             if enemy.name == "Clyde":
138                 clyde = enemy
139         # Cerca quali sono i dots/pellets più vicini al player
140         nearest_dot = dots[0]
141         for i, dot in enumerate(dots):
142             if ((nearest_dot.x-self.grid_pos.x)**2+(nearest_dot.y-self.grid_pos.y)
        **2) > ((dot.x-self.grid_pos.x)**2+(dot.y-self.grid_pos.y)**2):
143                 nearest_dot = dot
144         deltas_f = []
145         directions = [vec(0, -1), vec(-1, 0), vec(0, 1), vec(1, 0)]
146         # Per ogni direzione (dx, dy) calcolo la funzione delta(dx, dy)
147         for direction in directions:
```

```
148             if self.can_move_certain_direction(self.grid_pos, direction) and self.
    direction != self.inverse(direction):
149                 delta = []
150                 delta_xy = []
151                 player_new_pos = self.grid_pos+direction
152                 for enemy in enemies:
153                     delta.append((self.grid_pos.x-enemy.grid_pos.x)**2+(self.grid_pos
    .y-enemy.grid_pos.y)**2)
154                     enemy_new_pos = enemy.grid_pos+self.simulate_enemy(enemy, clyde)
155                     delta_xy.append((player_new_pos.x-enemy_new_pos.x)**2+(
    player_new_pos.y-enemy_new_pos.y)**2)
156                 Delta = 0
157                 for value in delta_xy:
158                     Delta += value
159                 pi = ((self.grid_pos.x-nearest_dot.x)**2+(self.grid_pos.x-nearest_dot
    .x)**2)/((player_new_pos.x-nearest_dot.x)**2+(player_new_pos.y-nearest_dot.y)
    **2+1)
160                 delta_f = 0
161                 for i, enemy in enumerate(enemies):
162                     if enemy.state == "Chase" or enemy.state == "Scatter":
163                         delta_f += (delta_xy[i]-delta[i])*(Delta-delta_xy[i])
164                 deltas_f.append((delta_f/Delta)+pi)
165             else:
166                 deltas_f.append(-100000)
167         # Restituisco l'indice del delta maggiore
168         if self.time_to_move():
169             print("Posizione: "+str(self.grid_pos)+", direzione attuale: "+str(self.
    direction))
170             for i, direction in enumerate(directions):
171                 print("\tDirezione: "+str(direction)+", valore delta: "+str(deltas_f[
    i])+", controllo: "+str(self.direction != self.inverse(direction)))
172         return numpy.argmax(deltas_f)
173 
174 ########## ENEMIES SIMULATION FUNCTIONS ##########
175 
176     # Simula quale direzione deve prendere il fantasma usando la posizione target che
     deve raggiungere
177     # Restituisce la posizone in cui si troverebbe il fantasma
178     # Esclude la direzione da cui il fantasma proviene
179     def simulate_enemy(self, enemy, clyde):
180         distances = []
181         directions = [vec(0, -1), vec(-1, 0), vec(0, 1), vec(1, 0)]
182         if enemy.name == "Clyde":
183             target_pos = self.clyde_target(enemy)
184         if enemy.name == "Pinky":
185             target_pos = self.pinky_target(enemy)
186         if enemy.name == "Inky":
187             target_pos = self.inky_target(enemy, clyde)
188         if enemy.name == "Blinky":
189             target_pos = self.blinky_target(enemy)
190         # Controllo se il fantasma è al di fuori della zona di spawn oppure no
191         # Se sì considero come posizione effettuiva la sua posizione più la sua
    direzione
192         # Altrimenti considero la sua posizione effettiva
193         if (not enemy.outside) or (not enemy.can_move()):
194             actual_pos = enemy.grid_pos
195         else:
196             actual_pos = enemy.grid_pos+enemy.direction
197         # Controllo se il fantasma è al di fuori della zona di spawn oppure no
198         # Se sì considero come posizione effettuiva la sua posizione più la sua
    direzione
199         # Altrimenti considero la sua posizione effettiva
200         if (not enemy.outside) or (not enemy.can_move()):
201             actual_pos = enemy.grid_pos
202         else:
```

```
203             actual_pos = enemy.grid_pos+enemy.direction
204         for direction in directions:
205             # Controllo se si può muovere in una direzione, e se si‘ calcolo la
    distanza dalla casella target
206             if enemy.can_move_certain_direction(actual_pos, direction) and enemy.
    direction != (-1*direction):
207                 # Se il fantasma si trova nella casella (14, 6), considera la
    posizione (14, 33) per
208                 # Calcolare la distanza effettiva dalla casella target
209                 if (actual_pos == vec(14, 6)) and direction == enemy.app.crossroadL:
210                     next_pos = vec(14, 33)
211                 # Se il fantasma si trova nella casella (14, 21), considera la
    posizione (14, -6) per
212                 # Calcolare la distanza effettiva dalla casella target
213                 elif (actual_pos == vec(14, 21)) and direction == enemy.app.
    crossroadR:
214                     next_pos = vec(14, -6)
215                 else:
216                     next_pos = actual_pos+direction
217                 distances.append(round((next_pos.x-target_pos.x)**2+((next_pos.y-
    target_pos.y))**2))
218             else:
219                 distances.append(3000)
220         # Cerco l'indice dell'elemento con distanza minore, e lo uso per deicdere(0:)
221         m = numpy.argmin(distances)
222         if m == 0:
223             return vec(0, -1)
224         elif m == 1:
225             return vec(-1, 0)
226         elif m == 2:
227             return vec(0, 1)
228         elif m == 3:
229             return vec(1, 0)
230 
231     # Clyde cerca di raggiungere la posizione in cui si trova il player
232     def clyde_target(self, enemy):
233         if not enemy.outside:
234             target_pos = vec(13, 11)
235         else:
236             target_pos = self.grid_pos
237         return target_pos
238 
239     # Pinky punta 4 caselle in avanti rispetto a dove è il player
240     # Ne considera la direzione, e se il player guarda su, il fantasma guardera‘ 4
    caselle su e 4 a sinistra
241     # (Behaviour dovuto ad un bug nelle prive versioni arcade)
242     def pinky_target(self, enemy):
243         if not enemy.outside:
244             target_pos = vec(13, 11)
245         else:
246             target_pos = self.grid_pos+(self.direction*4)
247             if self.direction == vec(0, -1):
248                 target_pos += vec(-4, 0)
249         return target_pos
250 
251     # Inky consuidera una casella avanti al player, punta una freccia da quella
    casella verso Clyde, e poi la ruota di 180 gradi
252     # Ne considera la direzione, e se il player guarda su, il fantasma guardera‘ 4
    caselle su e 4 a sinistra
253     # (Behaviour dovuto ad un bug nelle prive versioni arcade)
254     def inky_target(self, enemy, clyde):
255         if not enemy.outside:
256             target_pos = vec(13, 11)
257         else:
258             cell_pos = self.grid_pos+self.direction*2
```

```
259             if self.direction == vec(0, -1):
260                 cell_pos += vec(-2, 0)
261             offset = cell_pos-clyde.grid_pos
262             target_pos = clyde.grid_pos+(2*offset)
263         return target_pos
264 
265     # Blinky si comporta come Clyde, ma se è troppo vicino al player (8 blocchi di
    raggio), punta alla casella (30, 0)
266     def blinky_target(self, enemy):
267         if not enemy.outside:
268             target_pos = vec(13, 11)
269         else:
270             if (enemy.grid_pos.x-self.grid_pos.x)**2+(enemy.grid_pos.x-self.grid_pos.
    y)**2 > 64:
271                 target_pos = self.grid_pos
272             else:
273                 target_pos = vec(30, 0)
274         return target_pos
275 
276 ########## HELPER FUNCTIONS ##########
277 
278     # Calcola il vettore inverso
279     def inverse(self, vector):
280         x = vector.x
281         y = vector.y
282         if x == 1 or x == -1:
283             x *= -1
284         if y == 1 or y == -1:
285             y *= -1
286         return vec(x, y)
287 
288     # Calcola la posizione in cui deve disegnare il player
289     def get_pix_pos(self):
290         return vec((self.grid_pos[0]*self.app.cell_width)+TOP_BOTTOM_BUFFER//2+self.
    app.cell_width//2,
291                    (self.grid_pos[1]*self.app.cell_height) +
292                    TOP_BOTTOM_BUFFER//2+self.app.cell_height//2)
293 
294         print(self.grid_pos, self.pix_pos)
295 
296     # Controlla se il player può muoversi
297     # (controlla se si trova al centro della casella della griglia)
298     def time_to_move(self):
299         if int(self.pix_pos.x+TOP_BOTTOM_BUFFER//2) % self.app.cell_width == 0:
300             if self.direction == vec(1, 0) or self.direction == vec(-1, 0) or self.
    direction == vec(0, 0):
301                 return True
302         if int(self.pix_pos.y+TOP_BOTTOM_BUFFER//2) % self.app.cell_height == 0:
303             if self.direction == vec(0, 1) or self.direction == vec(0, -1) or self.
    direction == vec(0, 0):
304                 return True
305 
306     # Controlla se il fantasma può continuare a muoversi in una certa direzione
307     def can_move(self):
308         if vec(self.grid_pos+self.direction) in self.app.walls:
309             return False
310         return True
311 
312     # Controlla se il fantasma si può muovere in una certa direzione
313     def can_move_certain_direction(self, position, direction):
314         if vec(position+direction) in self.app.walls:
315             return False
316         return True
```

C.2 File "app_class.py"

```
import pygame
import sys
import copy
import time
from settings import *
from player_class import *
from enemy_class import *

vec = pygame.math.Vector2

class App:
    # Costruttore
    def __init__(self):
        pygame.init()
        self.screen = pygame.display.set_mode((WIDTH, HEIGHT))
        self.clock = pygame.time.Clock()
        self.running = True
        self.state = 'start'
        self.cell_width = MAZE_WIDTH//28
        self.cell_height = MAZE_HEIGHT//31
        self.walls = []
        self.barrier = []
        self.dots = []
        self.pellets = []
        self.crossroads = []
        self.crossroad_L = None
        self.crossroad_R = None
        self.total_dots = 0
        self.enemies = []
        self.enemies_names = ["Clyde", "Pinky", "Inky", "Blinky"]
        self.load()
        self.player = Player(self)
        self.make_enemies()

    # Funzione run: definisce cosa deve fare il gioco in base allo stato in cui si
    trova
    def run(self):
        while self.running:
            self.clock.tick(FPS)
            if self.state == "start":
                self.start_events()
                self.start_update()
                self.start_draw()
            elif self.state == "playing":
                self.playing_events()
                self.playing_update()
                self.playing_draw()
            elif self.state == "game over":
                self.game_over_events()
                self.game_over_update()
                self.game_over_draw()
            elif self.state == "victory":
                self.victory_events()
                self.victory_update()
                self.victory_draw()
            else:
                self.running = False
        pygame.quit()
        sys.exit()

########## HELPER FUNCTIONS ##########

```

```
62      # Disegna il testo sullo schermo nella posizione indicata
63      # Viene usato il colore e il font passati come parametri
64      def draw_text(self, words, screen, pos, size, colour, font_name, centered=False):
65          font = pygame.font.SysFont(font_name, size)
66          text = font.render(words, False, colour)
67          text_size = text.get_size()
68          if centered:
69              pos[0] = pos[0]-text_size[0]//2
70              pos[1] = pos[1]-text_size[1]//2
71          screen.blit(text, pos)
72
73      # Carica l'immagine "maze.png" e legge il file "walls.txt"
74      # Traduce il file "walls.txt" in informazioni riguardo il labirinto
75      def load(self):
76          self.background = pygame.image.load("maze.png")
77          self.background = pygame.transform.scale(self.background, (MAZE_WIDTH,
    MAZE_HEIGHT))
78          # Identifica i punti importanti della mappa
79          # (pareti, barriera, incroci, dots, pellets)
80          with open("walls.txt", "r") as file:
81              for yidx, line in enumerate(file):
82                  for xidx, char in enumerate(line):
83                      if char in ["1", "B"]:
84                          self.walls.append(vec(xidx, yidx))
85                          if char == "B":
86                              self.barrier.append(vec(xidx, yidx))
87                      elif char in ["D", "X", "L", "R", "P"]:
88                          self.total_dots += 1
89                          if char == "P":
90                              self.pellets.append(vec(xidx, yidx))
91                          else:
92                              self.dots.append(vec(xidx, yidx))
93                              if char == "X":
94                                  self.crossroads.append(vec(xidx, yidx))
95                              elif char == "L":
96                                  self.crossroad_L = vec(xidx, yidx)
97                              elif char == "R":
98                                  self.crossroad_R = vec(xidx, yidx)
99                      elif char == "Y":
100                         self.crossroads.append(vec(xidx, yidx))
101
102     # Crea gli oggetti "enemies" (i fantasmi)
103     def make_enemies(self):
104         for ind, name in enumerate(self.enemies_names):
105             # Se il fantasma si chiama "Clyde" allora si trova all'esterno della
106             # Zona di spawn, altrimenti si trova all'intero della zona di spawn
107             if name == "Clyde":
108                 self.enemies.append(Enemy(self, name, ind, True))
109             else:
110                 self.enemies.append(Enemy(self, name, ind, False))
111
112     # Resetta il gioco
113     def reset(self):
114         self.player = None
115         self.enemies = []
116         self.player = Player(self)
117         self.make_enemies()
118         self.load()
119         self.state = "playing"
120
121 ########## INTRO FUNCTIONS ##########
122
123     def start_events(self):
124         for event in pygame.event.get():
125             if event.type == pygame.QUIT:
```

```
126                 self.running = False
127             if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
128                 self.state = "playing"
129 
130     def start_update(self):
131         pass
132 
133     def start_draw(self):
134         self.screen.fill(BLACK)
135         self.draw_text("PUSH SPACE BAR", self.screen, [
136                        WIDTH//2, HEIGHT//2-50], START_TEXT_SIZE, (170, 132, 58),
    START_FONT, centered=True)
137         self.draw_text("1 PLAYER ONLY", self.screen, [
138                        WIDTH//2, HEIGHT//2+50], START_TEXT_SIZE, (44, 167, 198),
    START_FONT, centered=True)
139         self.draw_text("HIGH SCORE", self.screen, [4, 0],
140                        START_TEXT_SIZE, (255, 255, 255), START_FONT)
141         pygame.display.update()
142 
143 ########## PLAYING FUNCTIONS ##########
144 
145     def playing_events(self):
146         for event in pygame.event.get():
147             if event.type == pygame.QUIT:
148                 self.running = False
149 
150     def playing_update(self):
151         # Effettua l'udpate del player
152         value = self.player.calculate_direction(self.enemies, self.dots)
153         if value == 0:
154             self.player.move(vec(0, -1))
155         if value == 1:
156             self.player.move(vec(-1, 0))
157         if value == 2:
158             self.player.move(vec(0, 1))
159         if value == 3:
160             self.player.move(vec(1, 0))
161         self.player.update()
162         # Effettua lúpdate dei fantasmi
163         for enemy in self.enemies:
164             # Se il fantasma si chiama Inky e sono stati mangiati meno di 30 dots
165             # Allora il fantasma non può essere aggiornato
166             if enemy.name == "Inky" and self.player.eaten_dots < 30:
167                 enemy.update_state_only()
168             # Se il fantasma si chiama Inky e sono stati mangiati meno di 1/3 dei
    dots
169             # Allora il fantasma non può essere aggiornato
170             elif enemy.name == "Blinky" and self.player.eaten_dots < self.total_dots
    //3:
171                 enemy.update_state_only()
172             else:
173                 enemy.update()
174             # Se il fantasma ènella stessa casella del player
175             if self.player.grid_pos == enemy.grid_pos:
176                 # Se il fantasma è nello stato di "Chase"
177                 # Allora viene tolta una vita al player
178                 if enemy.state == "Chase":
179                     self.remove_life()
180                 # Se il fantasma è nello stato di "Frightened"
181                 # Allora il fantasma viene mangiato
182                 elif enemy.state == "Frightened":
183                     self.player.eat_enemy(enemy)
184         # Se il player ha mangiato tutti i dots, allora il player ha vinto
185         if self.player.eaten_dots == self.total_dots:
186             self.state = "victory"
```

```
187
188     def playing_draw(self):
189         self.screen.fill(BLACK)
190         self.screen.blit(self.background, (TOP_BOTTOM_BUFFER//2, TOP_BOTTOM_BUFFER
    //2))
191         self.draw_dots()
192         self.draw_pellets()
193         self.draw_text("CURRENT SCORE: {}".format(self.player.current_score),
194                        self.screen, [60, 0], 18, WHITE, START_FONT)
195         self.player.draw()
196         for enemy in self.enemies:
197             enemy.draw()
198         pygame.display.update()
199
200     # Rimuove una vita al player e riposiziona player e fantasmi
201     # Se le vite del player vanno a 0, allora il giocatore perdfe
202     def remove_life(self):
203         self.player.lives -= 1
204         if self.player.lives == 0:
205             self.state = "game over"
206         else:
207             self.player.grid_pos = PLAYER_START_POS
208             self.player.pix_pos = self.player.get_pix_pos()
209             self.player.direction *= 0
210             for enemy in self.enemies:
211                 enemy.grid_pos = vec(enemy.starting_pos)
212                 enemy.pix_pos = enemy.get_pix_pos()
213                 enemy.direction *= 0
214
215     def draw_dots(self):
216         for dot in self.dots:
217             pygame.draw.circle(self.screen, DOT_PELLET_COLOUR,
218                                (int(dot.x*self.cell_width)+self.cell_width//2+
    TOP_BOTTOM_BUFFER//2,
219                                 int(dot.y*self.cell_height)+self.cell_height//2+
    TOP_BOTTOM_BUFFER//2), 2)
220
221     def draw_pellets(self):
222         for pellet in self.pellets:
223             pygame.draw.circle(self.screen, DOT_PELLET_COLOUR,
224                                (int(pellet.x*self.cell_width)+self.cell_width//2+
    TOP_BOTTOM_BUFFER//2,
225                                 int(pellet.y*self.cell_height)+self.cell_height//2+
    TOP_BOTTOM_BUFFER//2), 4)
226
227 ########## GAME OVER FUNCTIONS ##########
228
229     def game_over_events(self):
230         for event in pygame.event.get():
231             if event.type == pygame.QUIT:
232                 self.running = False
233             if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
234                 self.reset()
235             if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE:
236                 self.running = False
237
238     def game_over_update(self):
239         pass
240
241     def game_over_draw(self):
242         self.screen.fill(BLACK)
243         quit_text = "Press the escape button to QUIT"
244         again_text = "Press SPACE bar to PLAY AGAIN"
245         self.draw_text("GAME OVER", self.screen, [WIDTH//2, 100],  52, RED, "arial",
    centered=True)
```

```
246         self.draw_text(again_text, self.screen, [
247                        WIDTH//2, HEIGHT//2],  36, (190, 190, 190), "arial", centered=
    True)
248         self.draw_text(quit_text, self.screen, [
249                        WIDTH//2, HEIGHT//1.5],  36, (190, 190, 190), "arial",
    centered=True)
250         pygame.display.update()
251
252 ########## VICTORY FUNCTIONS ##########
253
254     def victory_events(self):
255         for event in pygame.event.get():
256             if event.type == pygame.QUIT:
257                 self.running = False
258             if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
259                 self.reset()
260             if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE:
261                 self.running = False
262
263     def victory_update(self):
264         pass
265
266     def victory_draw(self):
267         self.screen.fill(BLACK)
268         quit_text = "Press the escape button to QUIT"
269         again_text = "Press SPACE bar to PLAY AGAIN"
270         self.draw_text("VICTORY!", self.screen, [WIDTH//2, 100],  52, VICTORY_COLOUR,
     "arial", centered=True)
271         self.draw_text(again_text, self.screen, [
272                        WIDTH//2, HEIGHT//2],  36, (190, 190, 190), "arial", centered=
    True)
273         self.draw_text(quit_text, self.screen, [
274                        WIDTH//2, HEIGHT//1.5],  36, (190, 190, 190), "arial",
    centered=True)
275         pygame.display.update()
```

Appendice D

Codice dell'implementazione dell'agente con DQN

Di seguito sono riportati i codici sorgente dell'implementazione dell'agente con DQN. Per evitare ripetizioni, sono mostrati i codici dei soli file che hanno subito modifiche. per i restanti, si fa riferimento ai codici mostrati nell'appendice B.

D.1 File "settings.py"

```
from pygame.math import Vector2 as vec

# Screen settings
WIDTH, HEIGHT = 610, 670
FPS = 60
TOP_BOTTOM_BUFFER = 50
MAZE_WIDTH, MAZE_HEIGHT = WIDTH-TOP_BOTTOM_BUFFER, HEIGHT-TOP_BOTTOM_BUFFER

# Colour settings
BLACK = (0, 0, 0)
RED = (255, 0, 0)
WHITE = (255, 255, 255)
GREY = (60, 60, 60)
BLUE = (0, 0, 255)

CLYDE_C = (255, 0, 0)
PINKY_C = (255, 184, 255)
INKY_C = (0, 255, 255)
BLINKY_C = (255, 184, 82)

PLAYER_COLOUR = (255, 255, 0)
DOT_PELLET_COLOUR = (255, 185, 175)

VICTORY_COLOUR = (204, 164, 61)

# Font settings
TITLE_TEXT_SIZE = 48

START_TEXT_SIZE = 16
START_FONT = 'arial black'

```

```
# Player settings
PLAYER_START_POS = vec(13.5, 23)
PLAYER_SPEED = 1.1
PLAYER_LIFES = 3

# Enemies settings
CLYDE_START_POS = vec(13.5, 11)
PINKY_START_POS = vec(13.5, 14)
INKY_START_POS = vec(12, 14)
BLINKY_START_POS = vec(15, 14)
ENEMIES_SPEED = 0.8

# Points settings
DOT_PTS = 10
PELLET_PTS = 50
VULNERABLE_GHOST_PTS = 200
CHERRY_PTS = 100
STRAWBERRY_PTS = 300
ORANGE_PTS = 500
APPLE_PTS = 700
MELON_PTS = 1000
GALAXIAN_BOSS_PTS = 2000
BELL_PTS = 3000
KEY_PTS = 5000

# Help settings
GRID = True
PLAYER_POS_CELL = True
ENEMY_POS_CELL = True

# Learning settings
EPOCHS = 500
EPOCH_SIZE = 5

# Neural network settings
LEARNING_RATE = 0.0002
GAMMA = 0.95
EPSILON = 1.0
INPUT_DIM = 26
N_ACTIONS = 4
ACTION_SPACE = [i for i in range(N_ACTIONS)]
EPS_MIN = 0.01
EPS_DECREASE = 0.001
EPS_DECAY = 0.001
HIDDEN_1_DIM = 50
HIDDEN_2_DIM = 50
```

D.2 File "mm_class.py"

```
import numpy as np
from recordtype import recordtype
from settings import *

Reward = recordtype("Reward", "value initial_value elapsed_time")

class MeasuresMemory:
    def __init__(self, file_all, file_avg):
        self.size = EPOCH_SIZE
        self.score = np.zeros((5, 1))
        self.lifes_lost = np.zeros((5, 1))
        self.moves = np.zeros((5, 1))
        self.counter = 0
        self.file_all = open(file_all, "a+")
```

```
        self.file_all.truncate(0)
        self.file_avg = open(file_avg, "a+")
        self.file_avg.truncate(0)

    def add_measure(self, score, lifes_lost, moves, write):
        self.score[self.counter, 0] = score
        self.lifes_lost[self.counter, 0] = lifes_lost
        self.moves[self.counter, 0] = moves
        self.counter = (self.counter + 1) % self.size
        if write:
            string = f"{score},{lifes_lost},{moves}\n"
            self.file_all.write(string)

    def average_values(self, write):
        avg_score = np.sum(self.score) / 5
        avg_lifes_lost = np.sum(self.lifes_lost) / 5
        avg_moves = np.sum(self.moves) / 5
        if write:
            string = f"{avg_score},{avg_lifes_lost},{avg_moves}\n"
            self.file_avg.write(string)
        return avg_score, avg_lifes_lost, avg_moves

    def close(self):
        self.file_all.close()
        self.file_avg.close()
```

D.3 File "rm_class.py"

```
import numpy as np
from rl_class import *
from settings import *

class ReplayMemory(object):

    def __init__(self, capacity, max_ticks):
        self.capacity = capacity
        self.reward_list = RewardList(max_ticks)
        self.reset(capacity, max_ticks)

    def push(self, state, action, next_state):
        if self.size < self.capacity:
            self.size += 1
        self.memory_state[self.position] = state
        self.memory_action[self.position] = action
        self.memory_next_state[self.position] = next_state
        self.memory_reward[self.position] = self.reward_list.rewards_sum()
        self.position = (self.position + 1) % self.capacity

    def get_memory_state(self): return self.memory_state[:self.size, :]

    def get_memory_action(self): return self.memory_action[:self.size]

    def get_memory_next_state(self): return self.memory_next_state[:self.size, :]

    def get_memory_reward(self): return self.memory_reward[:self.size]

    def set_memory_reward(self, position, reward, clear):
        if clear: self.memory_reward[position] = reward
        else: self.memory_reward[position] += reward

    def get_position_act(self): return self.position

    def get_position_prev(self): return (self.position - 1) % self.capacity
```

```
36
37      def get_position_succ(self): return (self.position + 1) % self.capacity
38
39      def add_reward(self, reward): self.reward_list.add_reward(reward)
40
41      def update_reward_list(self): self.reward_list.update()
42
43      def reset(self, capacity, max_ticks):
44          self.capacity = capacity
45          self.clear()
46          self.reward_list.reset(max_ticks)
47
48      def clear(self):
49          self.memory_state = np.zeros((self.capacity, INPUT_DIM), dtype=np.float32)
50          self.memory_action = np.zeros(self.capacity, dtype=np.int32)
51          self.memory_next_state = np.zeros((self.capacity, INPUT_DIM), dtype=np.
    float32)
52          self.memory_reward = np.zeros(self.capacity, dtype=np.float32)
53          self.size = 0
54          self.position = 0
55          self.reward_list.clear()
56
57      def update_reward_list(self): return self.reward_list.update()
58
59      def show(self):
60          print(f"State: {self.memory_state[:self.size, :]}\n",
61                f"Action: {self.memory_action[:self.size]}\n",
62                f"Next state: {self.memory_next_state[:self.size, :]}\n",
63                f"Reward: {self.memory_reward[:self.size]}\n")
```

D.4 File "nn_class.py"

```
1  import random
2  import copy
3  import torch
4  import torch.nn as nn
5  import torch.nn.functional as F
6  import torch.optim as optim
7  import numpy as np
8  from settings import *
9  from rm_class import *
10
11 import pprint
12
13 class DQN(nn.Module):
14     def __init__(self, lr, input_dim, hidden_1_dim, hidden_2_dim, n_actions):
15         super(DQN, self).__init__()
16         self.input_dim = input_dim
17         self.hidden_1_dim = HIDDEN_1_DIM
18         self.hidden_2_dim = HIDDEN_2_DIM
19         self.n_actions = n_actions
20         self.link_1 = nn.Linear(*self.input_dim, self.hidden_1_dim)
21         self.link_2 = nn.Linear(self.hidden_1_dim, self.hidden_2_dim)
22         self.link_3 = nn.Linear(self.hidden_2_dim, n_actions)
23         self.optimizer = optim.SGD(self.parameters(), lr=lr)
24         self.loss = nn.MSELoss()
25         self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
26         self.to(self.device)
27
28     def forward(self, x):
29         x = F.relu(self.link_1(x))
30         x = F.relu(self.link_2(x))
31         x = self.link_3(x)
```

```
32         return x
```

D.5 File "player_class.py"

```
1  import pygame
2  import time
3  import copy
4  import torch
5  import math
6  import numpy as np
7  from settings import *
8  from nn_class import *
9  from rm_class import *
10
11 vec = pygame.math.Vector2
12
13 class Player:
14     # Costruttore
15     def __init__(self, app, cycle):
16         self.app = app
17         self.neural_network = DQN(LEARNING_RATE, [INPUT_DIM], 30, 30, N_ACTIONS)
18         self.device = self.neural_network.device
19         self.reset(cycle)
20
21     def reset(self, cycle):
22         self.grid_pos = copy.deepcopy(PLAYER_START_POS)
23         self.starting_pos = copy.deepcopy(PLAYER_START_POS)
24         self.pix_pos = self.get_pix_pos()
25         self.direction = vec(1, 0)
26         self.stored_direction = None
27         self.able_to_move = True
28         self.current_score = 0
29         self.eaten_dots = 0
30         self.speed = 1
31         self.lifes = copy.deepcopy(PLAYER_LIFES)
32         self.counter = 1
33         self.old_state = None
34         self.old_action = None
35         self.old_grid_pos = None
36         self.epsilon = EPSILON - (cycle - 1) * EPS_DECREASE
37         size = 5 + math.floor(math.log(cycle, 5) * math.log(cycle, 10))
38         self.replay_memory = ReplayMemory(size)
39
40     # Update
41     def update(self):
42         # Se il player può muoversi, allora effettua un moviemnto
43         if self.able_to_move:
44             self.pix_pos += self.direction*self.speed
45         # Se si trova al centro della casella, e pertanto può cambiare la sua
       direzione
46         if self.time_to_move():
47             # Cambia la direzione
48             if self.stored_direction != None:
49                 self.direction = self.stored_direction
50             # Controlla se può continuare a muoversi
51             self.able_to_move = self.can_move()
52         # Controlla se ha preso uno dei due corridoi che porta dall'altra parte della
        mappa
53         if self.grid_pos == vec(28, 14) and self.stored_direction == vec(1, 0):
54             self.grid_pos = vec(0, 14)
55             self.pix_pos[0] = (self.grid_pos[0]-1)*self.app.cell_width+
       TOP_BOTTOM_BUFFER
56         if self.grid_pos == vec(-1, 14) and self.stored_direction == vec(-1, 0):
```

```
57              self.grid_pos = vec(27, 14)
58              self.pix_pos[0] = (self.grid_pos[0]-1)*self.app.cell_width+
    TOP_BOTTOM_BUFFER
59          # Calcola la posizione del fantasma nella griglia
60          self.grid_pos[0] = (self.pix_pos[0]-TOP_BOTTOM_BUFFER +
61                              self.app.cell_width//2)//self.app.cell_width+1
62          self.grid_pos[1] = (self.pix_pos[1]-TOP_BOTTOM_BUFFER +
63                              self.app.cell_height//2)//self.app.cell_height+1
64
65      # Funzione che disegna il player sullo schermo
66      # La seconda draw disegna le vite di cui dispone il player
67      def draw(self):
68          pygame.draw.circle(self.app.screen, PLAYER_COLOUR, (int(self.pix_pos.x), int(
    self.pix_pos.y)), self.app.cell_width//2-2)
69          for x in range(self.lifes):
70              pygame.draw.circle(self.app.screen, PLAYER_COLOUR, (30 + 20*x, HEIGHT -
    15), 7)
71
72      # Mangia il dot su cui si trova
73      # (rimuove il pellet dall'array e incrementa il numero di dot mangiati)
74      def eat_dot(self):
75          self.current_score += DOT_PTS
76          self.eaten_dots += 1
77
78      # Mangia il pellet su cui si trova (rimuove il pellet dall'array e incrementa il
    numero di dot mangiati)
79      # Inoltre setta tutti i fantasmi sullo stato di "Frightened"
80      def eat_pellet(self):
81          self.current_score += PELLET_PTS
82          self.eaten_dots += 1
83
84      # Mangia il fantasma concui si incrocia, e setta il fantasma nello stato di "
    Eaten"
85      # Inoltre incrementa il numero di fantasmi mangiati e calcola il percorso per
    arrivare alla zona di spawn
86      def eat_enemy(self, enemy):
87          self.current_score += VULNERABLE_GHOST_PTS*self.counter
88          self.counter += 1
89          enemy.enter_eaten_state()
90
91      # Salva la direzione appena ricevuta
92      def move(self, direction):
93          self.stored_direction = direction
94
95  ########## HELPER FUNCTIONS ##########
96
97      # Calcola la posizione in cui deve disegnare il player
98      def get_pix_pos(self):
99          return vec((self.grid_pos[0]*self.app.cell_width)+TOP_BOTTOM_BUFFER//2+self.
    app.cell_width//2,
100                     (self.grid_pos[1]*self.app.cell_height) +
101                     TOP_BOTTOM_BUFFER//2+self.app.cell_height//2)
102
103     # Controlla se il player può muoversi
104     # (controlla se si trova al centro della casella della griglia)
105     def time_to_move(self):
106         if int(self.pix_pos.x+TOP_BOTTOM_BUFFER//2) % self.app.cell_width == 0:
107             if self.direction == vec(1, 0) or self.direction == vec(-1, 0) or self.
    direction == vec(0, 0):
108                 return True
109         if int(self.pix_pos.y+TOP_BOTTOM_BUFFER//2) % self.app.cell_height == 0:
110             if self.direction == vec(0, 1) or self.direction == vec(0, -1) or self.
    direction == vec(0, 0):
111                 return True
112
```

```
113     # Controlla se il fantasma può continuare a muoversi in una certa direzione
114     def can_move(self):
115         if vec(self.grid_pos+self.direction) in self.app.walls:
116             return False
117         return True
118 
119     # Controlla se il player si può muovere in una certa direzione
120     def can_move_certain_direction(self, direction):
121         if vec(self.grid_pos+direction) in self.app.walls:
122             return False
123         return True
124 
125 ########## LEARNING FUNCTIONS ##########
126 
127     # Descrizione dello stato attuale
128     # (ritorna sia un ndarray sia che una tupla)
129     def get_actual_state(self, enemies, dots):
130         state = []
131         # Posizione (x, y) del player
132         state.append(self.grid_pos.x)
133         state.append(self.grid_pos.y)
134         # Per ogni fantasma, salvo la stato definito da
135         # distanza di Manhattan tra player e fantasma e stato
136         for idx, enemy in enumerate(enemies):
137             state.append(enemy.grid_pos.x)
138             state.append(enemy.grid_pos.y)
139             state.append(enemy.get_state_int())
140             state.append(-enemy.get_state_int()*enemy.get_manhattan_distance(self.
    grid_pos.x, self.grid_pos.y))
141         # Cerco il dot/pellet più vicino al player (considerata la posizione passata)
142         nearest_dot = dots[0]
143         for i, dot in enumerate(dots):
144             if ((nearest_dot.x-self.grid_pos.x)**2+(nearest_dot.y-self.grid_pos.y)
    **2) > ((dot.x-self.grid_pos.x)**2+(dot.y-self.grid_pos.y)**2):
145                 nearest_dot = dot
146         state.append(nearest_dot.x)
147         state.append(nearest_dot.y)
148         # Presenza di un muro nelle direzioni possibili
149         for direction in [vec(0, -1), vec(-1, 0), vec(0, 1), vec(1, 0)]:
150             state.append(1) if not self.can_move_certain_direction(direction) else
    state.append(0)
151         # Numero di vite rimanenti e di dot rimanenti
152         state.append(self.lifes)
153         state.append(len(dots))
154         return state, np.array(state)
155 
156     # Decisione della prossima mossa
157     def calculate_direction(self, enemies, dots):
158         action = None
159         if self.time_to_move():
160             input_array, state = self.get_actual_state(enemies, dots)
161             # Se ottengo un numero randomico superiore all'epsilon salvato nella rete
     neurale
162             if np.random.random() > self.epsilon:
163                 # Creazione del tensor
164                 input_layer = torch.tensor(input_array).to(self.neural_network.device
    )
165                 #print(f"Input player: {input_layer}")
166                 # Calcolo del tesnor di output
167                 output_layer = self.neural_network(input_layer)
168                 #print(f"Output layer: {output_layer}")
169                 # Calcolo della posizione del valore più alto
170                 action = torch.argmax(output_layer).item()
171                 #print(f"Azione: {action}")
172             # Altrimenti
```

```
173             else:
174                 # Scegli un'azione casiale tra quelle disponibili
175                 action = np.random.choice(ACTION_SPACE)
176         return state, action
177 
178     # Salvo la transizione in memoria solo se mi trovo al centro di una
179     # nuova casella (e aggiorno la reward list)
180     def save_tuple(self, state, action, reward):
181         if self.grid_pos != self.old_grid_pos:
182             self.save_transition(state, action, reward)
183             self.old_grid_pos = copy.deepcopy(self.grid_pos)
184 
185     # Effettua il learning
186     # Se clear è settato su true, pulisce la replay memory, altrimenti aggiorna la
    reward list
187     def learn(self, clear):
188         if self.replay_memory.size > 0:
189             self.neural_network.optimizer.zero_grad()
190             # Recupera l'index del batch della replay memory
191             batch_index = np.arange(self.replay_memory.size, dtype=np.int32)
192             # ottine il batch degli stati salvati nella replay memory
193             state_array = self.replay_memory.get_memory_state()
194             state_batch = torch.tensor(state_array).to(self.device)
195             # ottine il batch degli stati successivi salvati nella replay memory
196             next_state_array = self.replay_memory.get_memory_next_state()
197             next_state_batch = torch.tensor(next_state_array).to(self.device)
198             # ottine il batch della zioni salvate nella replay memory
199             action_batch = self.replay_memory.get_memory_action()
200             # ottine il batch delle reward salvate nella replay memory
201             reward_array = self.replay_memory.get_memory_reward()
202             reward_batch = torch.tensor(reward_array).to(self.device)
203             # Calcola il Q-value dello stato attuale
204             q_eval = self.neural_network.forward(state_batch)
205             q_target = q_eval[batch_index, action_batch]
206             # Calcola il Q-value dello stato successivo
207             q_next = self.neural_network.forward(next_state_batch)
208             # Calcola il Q-value target usando il Q-value massimo dello stato
    successivo e la reward
209             q_predict = reward_batch + GAMMA * torch.max(q_next, dim=1)[0]
210             # Calcola la loss tra il Q-value dello stato attuale e il Q-value target
211             # Propaga la loss all'indietro
212             loss = self.neural_network.loss(q_predict, q_target).to(self.device)
213             #print(f"Loss: {loss}\nq_target: {q_target}\nq_predict: {q_predict}\
    nreward: {reward_batch}")
214             #print(f"Loss: {loss}")
215             loss.backward()
216             self.neural_network.optimizer.step()
217             # Effettua il decadimento di epsilon
218             self.epsilon = self.epsilon - EPS_DECAY if self.epsilon > EPS_MIN else
    copy.deepcopy(EPS_MIN)
219             # Se clear è uguale a true, ripulisce la replay memory
220             if clear: self.replay_memory.clear()
221 
222 ########## LEARNING HELP FUNCTIONS ##########
223 
224     # Salva la transizione
225     def save_transition(self, state, action, reward):
226         if self.old_state is None:
227             self.old_state = state
228             self.old_action = action
229         else:
230             self.replay_memory.push(self.old_state, self.old_action, state, reward)
231             self.old_state = state
232             self.old_action = action
233             self.reward = 0
```

```
234
235     # Ripulisco la memoria delle transizioni
236     def reset_replay_memory(self, capacity):
237         self.replay_memory.reset(capacity)
238         self.old_state = None
239         self.old_action = None
240
241     # Mostra il contenuto della mamoria delle transizioni
242     def show_replay_memory(self): self.replay_memory.show()
243
244     # Restituisce la dimensione della replay memory
245     def get_size_reward_memory(self): return self.replay_memory.size
246
247     # Restituisce la posizione del puntatore della replay memory
248     def get_position_reward_memory(self): return self.replay_memory.position
249
250     # Restituisce la capacità della replay memory
251     def get_capacity_reward_memory(self): return self.replay_memory.capacity
252
253     # Salva il modello della rete nuerale su file
254     def save_model(self): torch.save(self.neural_network, "neural_network.pt")
255
256     # Carica il modello checkpoint della rete neurale
257     def load_model(self, file):
258         self.neural_network.load_state_dict(torch.load(file, map_location=self.
    neural_network.device))
259         self.neural_network.eval()
```

D.6 File "app_class.py"

```
1  import pygame
2  import sys
3  import copy
4  import time
5  import numpy as np
6  from settings import *
7  from player_class import *
8  from enemy_class import *
9  from mm_class import *
10 from es_class import *
11
12 vec = pygame.math.Vector2
13
14 class App:
15     # Costruttore
16     def __init__(self):
17         pygame.init()
18         self.screen = pygame.display.set_mode((WIDTH, HEIGHT))
19         self.clock = pygame.time.Clock()
20         self.running = True
21         self.state = "start"
22         self.cell_width = MAZE_WIDTH//28
23         self.cell_height = MAZE_HEIGHT//31
24         self.enemies_names = ["Clyde", "Pinky", "Inky", "Blinky"]
25         self.load()
26         self.make_enemies()
27         self.cycle = 1
28         self.player = Player(self, self.cycle)
29         self.measueres_memory = MeasuresMemory("general_measures.txt", "
    average_measures.txt")
30         self.old_grid_pos = None
31         self.old_direction = self.player.direction
32         self.moves_done = 0
```

```
33          self.early_stopping = EarlyStopping(10)
34          self.prev_distances = []
35          self.prev_near_dot = []
36          self.reward_list = []
37          self.check_learn = False
38
39      # Funzione run: definisce cosa deve fare il gioco in base allo stato in cui si
        trova
40      def run(self):
41          while self.running:
42              self.clock.tick(FPS)
43              if self.state == "start":
44                  self.start_events()
45                  self.start_update()
46              elif self.state == "playing":
47                  self.playing_events()
48                  self.playing_update()
49                  self.playing_draw()
50              elif self.state == "game over":
51                  self.game_over_events()
52                  self.game_over_update()
53              elif self.state == "victory":
54                  self.victory_events()
55                  self.victory_update()
56              else:
57                  self.running = False
58          pygame.quit()
59          sys.exit()
60
61  ########## HELPER FUNCTIONS ##########
62
63      # Disegna il testo sullo schermo nella posizione indicata
64      # Viene usato il colore e il font passati come parametri
65      def draw_text(self, words, screen, pos, size, colour, font_name, centered=False):
66          font = pygame.font.SysFont(font_name, size)
67          text = font.render(words, False, colour)
68          text_size = text.get_size()
69          if centered:
70              pos[0] = pos[0]-text_size[0]//2
71              pos[1] = pos[1]-text_size[1]//2
72          screen.blit(text, pos)
73
74      # Carica l'immagine "maze.png" e legge il file "walls.txt"
75      # Traduce il file "walls.txt" in informazioni riguardo il labirinto
76      def load(self):
77          self.background = pygame.image.load("maze.png")
78          self.background = pygame.transform.scale(self.background, (MAZE_WIDTH,
        MAZE_HEIGHT))
79          # Identifica i punti importanti della mappa
80          # (pareti, barriera, incroci, dots, pellets)
81          self.walls = []
82          self.barrier = []
83          self.dots = []
84          self.pellets = []
85          self.crossroads = []
86          self.crossroad_L = None
87          self.crossroad_R = None
88          self.total_dots = 0
89          with open("walls.txt", "r") as file:
90              for yidx, line in enumerate(file):
91                  for xidx, char in enumerate(line):
92                      if char in ["1", "B"]:
93                          self.walls.append(vec(xidx, yidx))
94                          if char == "B":
95                              self.barrier.append(vec(xidx, yidx))
```

```
96                      elif char in ["D", "X", "L", "R", "P"]:
97                          self.total_dots += 1
98                          if char == "P":
99                              self.pellets.append(vec(xidx, yidx))
100                         else:
101                             self.dots.append(vec(xidx, yidx))
102                             if char == "X":
103                                 self.crossroads.append(vec(xidx, yidx))
104                             elif char == "L":
105                                 self.crossroad_L = vec(xidx, yidx)
106                             elif char == "R":
107                                 self.crossroad_R = vec(xidx, yidx)
108                     elif char == "Y":
109                         self.crossroads.append(vec(xidx, yidx))
110
111     # Crea gli oggetti "enemies" (i fantasmi)
112     def make_enemies(self):
113         self.enemies = []
114         for ind, name in enumerate(self.enemies_names):
115             # Se il fantasma si chiama "Clyde" allora si trova all'esterno della
116             # Zona di spawn, altrimenti si trova all'intero della zona di spawn
117             if name == "Clyde":
118                 self.enemies.append(Enemy(self, name, ind, True))
119             else:
120                 self.enemies.append(Enemy(self, name, ind, False))
121
122     # Resetta il gioco
123     def reset(self, cycle):
124         self.player.reset(cycle)
125         self.make_enemies()
126         self.load()
127         self.state = "playing"
128
129     # Funzione di segno
130     def sign(self, x):
131         if x >= 0: return +1
132         else: return -1
133
134     # Somma dei valori di una lista
135     def list_sum(self, l):
136         r = 0
137         for x in l:
138             r += x
139         return r
140
141 ########## INTRO FUNCTIONS ##########
142
143     def start_events(self):
144         self.state = "playing"
145
146     def start_update(self):
147         pass
148
149     def start_draw(self):
150         self.screen.fill(BLACK)
151         self.draw_text("PUSH SPACE BAR", self.screen, [
152                        WIDTH//2, HEIGHT//2-50], START_TEXT_SIZE, (170, 132, 58),
    START_FONT, centered=True)
153         self.draw_text("1 PLAYER ONLY", self.screen, [
154                        WIDTH//2, HEIGHT//2+50], START_TEXT_SIZE, (44, 167, 198),
    START_FONT, centered=True)
155         self.draw_text("HIGH SCORE", self.screen, [4, 0],
156                        START_TEXT_SIZE, (255, 255, 255), START_FONT)
157         pygame.display.update()
158
```

```
########## PLAYING FUNCTIONS ##########

    def playing_events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.running = False

    def playing_update(self):
        move_done = False
        clear = False
        state = []
        action = -1
        # Sceglie la mossa successiva solo se si trova al centro della casella
        if self.player.time_to_move():
            state, action = self.player.calculate_direction(self.enemies, self.dots)
            if action == 0:
                self.player.move(vec(0, -1))
            if action == 1:
                self.player.move(vec(-1, 0))
            if action == 2:
                self.player.move(vec(0, 1))
            if action == 3:
                self.player.move(vec(1, 0))
            move_done = True
        # Effettua l'udpate del player
        self.player.update()
        # Se non si può muovere nella direzione scelta, assegna come reward -1000
        if not self.player.can_move() and self.player.time_to_move():
            self.reward_list.append(-1000)
        # Se si trova su un dot, allora chiama la funzione per mangiare il dot
        if self.on_dot():
            self.dots.remove(self.player.grid_pos)
            self.player.eat_dot()
            self.reward_list.append(math.exp(2))
        # Se si trova su un pellet, allora chiama la funzione per mangiare un pellet
        if self.on_pellet():
            self.pellets.remove(self.player.grid_pos)
            self.player.eat_pellet()
            self.reward_list.append(math.exp(2))
            # Per ogni fantasma vado a settare il suo stato in "Frightened", a
    cambiargli colore e a decrementare la sua velocità
            for enemy in self.enemies:
                # Controllo se il fantasma è nello stato di "Chase"
                if enemy.state == "Chase":
                    enemy.enter_frightened_state(initial_time = time.clock())
        # Effettua l'update dei fantasmi
        for enemy in self.enemies:
            # Se il fantasma si chiama Inky e sono stati mangiati meno di 30 dots
            # Allora il fantasma non può essere aggiornato
            if enemy.name == "Inky" and self.player.eaten_dots < 30:
                enemy.update_state_only()
            # Se il fantasma si chiama Inky e sono stati mangiati meno di 1/3 dei
    dots
            # Allora il fantasma non può essere aggiornato
            elif enemy.name == "Blinky" and self.player.eaten_dots < self.total_dots
    //3:
                enemy.update_state_only()
            else:
                enemy.update()
            # Se il fantasma ènella stessa casella del player
            if self.player.grid_pos == enemy.grid_pos:
                # Se il fantasma è nello stato di "Chase"
                # Allora viene tolta una vita al player
                if enemy.state == "Chase":
                    self.remove_life()
```

```
                    clear = True
                # Se il fantasma è nello stato di "Frightened"
                # Allora il fantasma viene mangiato
                elif enemy.state == "Frightened":
                    self.player.eat_enemy(enemy)
                    self.reward_list.append(math.exp(4))
        # Se il player ha mangiato tutti i dots, allora il player ha vinto
        if self.player.eaten_dots == self.total_dots:
            self.state = "victory"
            self.reward_list.append(math.exp(5))
            clear = True
        # Incrementa il numero di mosse effettuate
        # Solo se la casella attuale è differente da quella successiva
        if self.old_grid_pos != self.player.grid_pos and self.old_grid_pos != None:
            self.moves_done += 1
            self.reward_list.append(-1)
        # Se si trova al centro della casella
        # Calcola il valore di correzione
        correction = 0
        if self.player.time_to_move():
            correction = self.calculate_correction_value()
        # Se l'agente ha effettuato un cambio di mossa Aggiunge la tupla
        # (stato, azione, nuovo stato, reward) attuale alla memoria e resetta la reward list
        if move_done:
            self.player.save_tuple(state, action, self.list_sum(self.reward_list) + correction)
            self.check_learn = False
        self.reward_list = []
        # Se la replay memory è piena e non siamo entrati in early stopping effettua l'apprendimento
        # (Caso speciale: se si ha perso una vita/la partita, si effettua lo stesso il learning)
        if not self.early_stopping.early_stop:
            if clear: self.player.learn(clear)
            else:
                size = self.player.get_size_reward_memory()
                position = self.player.get_position_reward_memory()
                capacity = self.player.get_capacity_reward_memory()
                if size == capacity and size != 0 and position == 0 and not self.check_learn:
                    self.player.learn(clear)
                    self.check_learn = True
        self.old_grid_pos = copy.deepcopy(self.player.grid_pos)

    # Calcola il valore di correzione
    def calculate_correction_value(self):
        correction = 0
        # Calcola la distanza tra i player e i fantasmi
        distances = [None] * 4
        for idx, enemy in enumerate(self.enemies):
            distances[idx] = abs(self.player.grid_pos.x-enemy.grid_pos.x)+abs(self.player.grid_pos.y-enemy.grid_pos.y) if enemy.outside else 60
        # Calcola le differenze tra le distanze attuali e quelle precedenti
        if len(self.prev_distances) != 0:
            differences = [None] * 4
            absolute = [None] * 4
            relative = [None] * 4
            correction = 0
            # Calcola i valori "assoluti"e "relativi" per il calcolo della correzione
            for idx, enemy in enumerate(self.enemies):
                differences[idx] = distances[idx] - self.prev_distances[idx]
                absolute[idx] = math.exp(6 - distances[idx]) #absolute[idx] = max(math.exp(6 - distances[idx]) - 1, 0)
                relative[idx] = self.sign(differences[idx]) * math.exp(-differences[
```

```
    idx])
279                 correction += enemy.get_state_int() * absolute[idx] * relative[idx]
280         self.prev_distances = distances
281         return correction
282 
283     def playing_draw(self):
284         self.screen.fill(BLACK)
285         self.screen.blit(self.background, (TOP_BOTTOM_BUFFER//2, TOP_BOTTOM_BUFFER
    //2))
286         self.draw_dots()
287         self.draw_pellets()
288         self.draw_text("CURRENT SCORE: {}".format(self.player.current_score),
289                        self.screen, [60, 0], 18, WHITE, START_FONT)
290         self.draw_text("CYCLE: {}".format(self.cycle),
291                        self.screen, [WIDTH-180, 0], 18, WHITE, START_FONT)
292         self.player.draw()
293         for enemy in self.enemies:
294             enemy.draw()
295         pygame.display.update()
296 
297     # Verifica se il player si trova si un dot
298     # (ossia se si trova al centro della casella dove si trova il dot)
299     def on_dot(self):
300         if self.player.grid_pos in self.dots:
301             if int(self.player.pix_pos.x+TOP_BOTTOM_BUFFER//2) % self.cell_width ==
    0:
302                 if self.player.direction == vec(1, 0) or self.player.direction == vec
    (-1, 0):
303                     return True
304             if int(self.player.pix_pos.y+TOP_BOTTOM_BUFFER//2) % self.cell_height ==
    0:
305                 if self.player.direction == vec(0, 1) or self.player.direction == vec
    (0, -1):
306                     return True
307         return False
308 
309     # Verifica se il player si trova si un pellet
310     # (ossia se si trova al centro della casella dove si trova il pellet)
311     def on_pellet(self):
312         if self.player.grid_pos in self.pellets:
313             if int(self.player.pix_pos.x+TOP_BOTTOM_BUFFER//2) % self.cell_width ==
    0:
314                 if self.player.direction == vec(1, 0) or self.player.direction == vec
    (-1, 0):
315                     return True
316             if int(self.player.pix_pos.y+TOP_BOTTOM_BUFFER//2) % self.cell_height ==
    0:
317                 if self.player.direction == vec(0, 1) or self.player.direction == vec
    (0, -1):
318                     return True
319         return False
320 
321     # Rimuove una vita al player e riposiziona player e fantasmi
322     # Se le vite del player vanno a 0, allora il giocatore perde
323     def remove_life(self):
324         self.player.lifes -= 1
325         if self.player.lifes == 0:
326             self.state = "game over"
327             self.reward_list.append(-math.exp(4))
328         else:
329             self.player.grid_pos = copy.deepcopy(self.player.starting_pos)
330             self.player.pix_pos = self.player.get_pix_pos()
331             self.player.direction = vec(1, 0)
332             for enemy in self.enemies:
333                 enemy.grid_pos = copy.deepcopy(vec(enemy.starting_pos))
```

```
334                 enemy.pix_pos = enemy.get_pix_pos()
335                 enemy.direction = vec(1, 0)
336                 if enemy.name == "Clyde":
337                     enemy.outside = True
338                 else:
339                     enemy.outside = False
340             self.reward_list.append(-math.exp(6.25))
341 
342     def draw_dots(self):
343         for dot in self.dots:
344             pygame.draw.circle(self.screen, DOT_PELLET_COLOUR,
345                                (int(dot.x*self.cell_width)+self.cell_width//2+
    TOP_BOTTOM_BUFFER//2,
346                                 int(dot.y*self.cell_height)+self.cell_height//2+
    TOP_BOTTOM_BUFFER//2), 2)
347 
348     def draw_pellets(self):
349         for pellet in self.pellets:
350             pygame.draw.circle(self.screen, DOT_PELLET_COLOUR,
351                                (int(pellet.x*self.cell_width)+self.cell_width//2+
    TOP_BOTTOM_BUFFER//2,
352                                 int(pellet.y*self.cell_height)+self.cell_height//2+
    TOP_BOTTOM_BUFFER//2), 4)
353 
354 ########## GAME OVER FUNCTIONS ##########
355 
356     def game_over_events(self):
357         self.measueres_memory.add_measure(self.player.current_score, PLAYER_LIFES -
    self.player.lifes, self.moves_done, True)
358         if self.cycle % EPOCH_SIZE == 0:
359             avg_score, avg_lifes_lost, avg_moves = self.measueres_memory.
    average_values(True)
360             self.early_stopping.__call__(avg_score, self.player.neural_network)
361             if self.early_stopping.early_stop:
362                 self.player.load_model(self.early_stopping.path)
363             print(f"EPOCH {self.cycle / EPOCH_SIZE}: score {avg_score}, lives lost {
    avg_lifes_lost}, moves done {avg_moves}")
364         self.cycle += 1
365         if math.ceil(self.cycle / EPOCH_SIZE) > EPOCHS:
366             self.measueres_memory.close()
367             self.player.save_model()
368             self.running = False
369         self.reset(self.cycle)
370         self.moves_done = 0
371 
372     def game_over_update(self):
373         pass
374 
375     def game_over_draw(self):
376         self.screen.fill(BLACK)
377         quit_text = "Press the escape button to QUIT"
378         again_text = "Press SPACE bar to PLAY AGAIN"
379         self.draw_text("GAME OVER", self.screen, [WIDTH//2, 100],  52, RED, "arial",
    centered=True)
380         self.draw_text(again_text, self.screen, [WIDTH//2, HEIGHT//2],  36, (190,
    190, 190), "arial", centered=True)
381         self.draw_text(quit_text, self.screen, [WIDTH//2, HEIGHT//1.5],  36, (190,
    190, 190), "arial", centered=True)
382         pygame.display.update()
383 
384 ########## VICTORY FUNCTIONS ##########
385 
386     def victory_events(self):
387         self.measueres_memory.add_measure(self.player.current_score, PLAYER_LIFES -
    self.player.lifes, self.moves_done, True)
```

```
388         if self.cycle % EPOCH_SIZE == 0:
389             self.early_stopping.__call__(avg_score, self.player.neural_network)
390             avg_score, avg_lifes_lost, avg_moves = self.measueres_memory.
    average_values(True)
391             if self.early_stopping.early_stop:
392                 self.player.load_model(self.early_stopping.path)
393             print(f"EPOCH {self.cycle / EPOCH_SIZE}: score {avg_score}, lives lost {
    avg_lifes_lost}, moves done {avg_moves}")
394         self.cycle += 1
395         if math.ceil(self.cycle / EPOCH_SIZE) > EPOCHS:
396             self.measueres_memory.close()
397             self.player.save_model()
398             self.running = False
399         self.reset(self.cycle)
400         self.moves_done = 0
401 
402     def victory_update(self):
403         pass
404 
405     def victory_draw(self):
406         self.screen.fill(BLACK)
407         quit_text = "Press the escape button to QUIT"
408         again_text = "Press SPACE bar to PLAY AGAIN"
409         self.draw_text("VICTORY!", self.screen, [WIDTH//2, 100],  52, VICTORY_COLOUR,
     "arial", centered=True)
410         self.draw_text(again_text, self.screen, [
411                         WIDTH//2, HEIGHT//2],  36, (190, 190, 190), "arial", centered=
    True)
412         self.draw_text(quit_text, self.screen, [
413                         WIDTH//2, HEIGHT//1.5],  36, (190, 190, 190), "arial",
    centered=True)
414         pygame.display.update()
```

Elenco delle figure

Bibliografia

[1] Stuart Russel and Peter Norvig; *"Intelligenza artificiale - Un approccio moderno"*; Pearson, 2010, terza edizione.

[2] Jamey Pittman; *"The Pac-Man Dossier"*; 2015.

[3] Chad Birch; *"Understanding Pac-Man Ghost Behaviour"*; 2010.

[4] Google DeepMing; *"AlphaGo"*.

[5] Google DeepMind; *"AlphaZero: Shedding new light on chess, shogi, and Go"*, 2010.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves Ioannis Antonoglou, Daan Wierstra and Martin Riedmiller; *"Playing Atari with Deep Reinforcement Learning"*; NIPS Deep Learning Workshop, 2013, università di Toronto.

[7] Sylvain Arlot and Alain Celisse; *"A survey of cross-validation procedures for model selection"*; Amer. Statist. Assoc., the Bernoulli Soc., the Inst. Math. Statist., and the Statist. Soc. Canada, 2010