Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

PROGETTAZIONE E SVILUPPO DI UN BOT AUTOMATICO PER GIOCHI DI STRATEGIA SU DEVICE MOBILI

Relatore:
Chiar.mo Prof.
ANDREA ASPERTI

Presentata da:
GABRIELE ORAZI

Correlatore:
Chiar.mo Dott.
LUCA BEDOGNI

Sessione III
Anno Accademico 2016/17

Ai miei genitori

Introduzione

In questa tesi viene discussa l'implementazione di un bot automatico in grado di interfacciarsi ad un device mobile, al fine di controllare autonomamente le fasi di battaglia di un gioco strategico.
In particolare sono stati presi in considerazione giochi di tipologia strategico-gestionale, dove il giocatore è tenuto ad amministrare il proprio villaggio, curando lo sviluppo delle strutture in esso contenute. Scopo del giocatore infatti è quello di potenziare al massimo gli edifici attraverso l'impiego di risorse, queste ultime ottenibili tramite battaglie online. Durante una sfida il giocatore si trova a dover attaccare e distruggere il villaggio di un altro utente per rubare quante più risorse possibili.
Il focus di questa tesi ricade proprio sulla fase offensiva, dove il bot implementato riesce a portare a termine battaglie autonomamente ed è potenzialmente in grado di interagire con qualsiasi dispositivo che utilizzi Android come sistema operativo. Il device in questione viene interamente controllato da un calcolatore esterno tramite l'invio di segnali atti ad emulare in tutto e per tutto quelli di un giocatore reale che sfiora lo schermo touchscreen del dispositivo.
In primo luogo il bot è in grado di acquisire le schermate di gioco dal device per poi trasferirle nel calcolatore connesso. Successivamente le immagini ottenute sono preprocessate con lo scopo di estrarre informazioni rilevanti, che verranno poi impiegate per elaborare una strategia di attacco valida. Il bot attua la tattica prodotta, traducendo e concretizzando quest'ultima tramite l'invio di una serie di gesture al device. In ultimo, a battaglia conclusa, il

bot è in grado di acquisire i risultati ottenuti.
Per questo progetto è stato scelto di utilizzare Python[1] come linguaggio di programmazione: la scelta è stata dettata dalla compatibilità con le librerie di interesse a questa tesi quali OpenCV[2], SciPy[3], NumPy[4] e Keras[5].
L'inclusione di Keras come framework per il machine learning è dovuta all'intento di sviluppare, oltre al bot di gioco, delle tecniche per ottimizzare la strategia di attacco. Questa parte è rimasta però largamente inesplorata per le ragioni tecniche che verranno ampiamente discusse in seguito.
Questa tesi si pone quindi all'intersezione tra diverse tematiche interessanti: notevole è innanzitutto il metodo con cui le tecniche di image processing, solitamente utilizzate per la classificazione[6] [7] o segmentazione[8] di immagini, in questo caso vengano impiegate per comandare un device mobile. In aggiunta di ciò, utilizzare tecniche di intelligenza artificiale per istruire un calcolatore ad impartire comandi ad un altro dispositivo esterno può rappresentare un caso di studio che apre a diversi possibili futuri sviluppi.

Da dove nasce l'idea

Il videogioco è una forma di intrattenimento che nell'ultimo decennio sta prendendo sempre più campo tra la concezione popolare, riscuotendo molto successo in fasce di pubblico a partire dai più giovani.
Grazie all'avvento degli smartphone però il mondo dal gaming è arrivato alla portata di tutti: attualmente una considerevole parte del mercato videoludico è interamente dedicata al *mobile gaming*.
La diffusione di questi dispositivi e la facilità di fruizione di tali tipologie di giochi hanno generato la nascita di intere software house e business unit di aziende già presenti nel mercato dell'intrattenimento elettronico e non.

Nell'ambito videoludico trova spazio anche un fiorente settore dell'informatica che negli ultimi tempi è diventato argomento di discussione comune e del quale anche i media iniziano a parlare assiduamente: l'intelligenza artificiale.

Quest'ultima viene da anni impiegata per generare dei comportamenti responsivi, adattivi e intelligenti per gli NPC[1].
Più recentemente però l'intelligenza artificiale inizia a trovare spazio anche sotto forma di bot intelligenti in grado di imparare a giocare con dei videogiochi e di apprendere gradualmente dagli errori commessi.
Sono state sviluppate moltissime sperimentazioni di bot toccando diverse tipologie di videogame. I giochi più frequentemente impiegati in questo campo sono gli arcade[9], in particolar modo gli Atari games[10], ma non mancano anche tipologie più complesse come ad esempio gli FPS[2] [11] [12] [13], fino ad arrivare anche ai giochi roguelike[3] [14].
A testimonianza del progresso continuo in atto in questo campo di studio, Elon Mask, famoso CEO e fondatore di realtà innovative quali Tesla Motors e SpaceX e cofondatore di PayPal, nell'estate del 2017 ha dichiarato che un game bot sviluppato da OpenAI[4] ha sconfitto uno dei più forti videogiocatori a livello mondiale[5].

L'idea di questa tesi nasce proprio dal connubio tra il mobile gaming e l'intelligenza artificiale applicata ai videogiochi. Lo scopo è quello di riuscire a creare uno strumento in grado di fruire dei contenuti di un'applicazione proprio come farebbe un reale utente, analizzando le schermate visualizzate e reagendo conseguentemente inviando segnali al dispositivo attraverso l'utilizzo di uno schermo touchscreen.

[1] Non-Player Character, personaggi che appaiono in un videogame non controllati da un giocatore

[2] First Person Shooter

[3] Videogiochi di ruolo caratterizzati da mappe casuali, movimenti a turno e morte permanente

[4] OpenAI è un'organizzazione non profit di ricerca sull'intelligenza artificiale cofondata da Elon Mask

[5] CNN Tech: http://cnnmon.ie/2uBjf7e

Struttura della tesi

In questo elaborato viene illustrata l'implementazione del progetto, le scelte adoperate in fase di sviluppo e le problematiche affrontate.
Nel primo capitolo vengono spiegate le motivazioni che hanno portato alla scelta del gioco e descritte le meccaniche di quest'ultimo.
Nel secondo capitolo si presenta il modo in cui è stato possibile comandare un device con sistema operativo Android per riuscire a fruire dell'applicazione scelta.
Nel terzo capitolo vengono trattati sia i passi necessari per processare l'immagine del villaggio avversario da attaccare in modo da piazzare efficacemente le truppe offensive, sia il salvataggio delle informazioni relative alle battaglie effettuate.
Il capitolo quarto descrive come sono stati rielaborati i dati raccolti per allenare una prima rete neurale in grado di prevedere l'esito degli scontri e le relative problematiche rilevate sulla base dei dati acquisiti.
Le conclusioni infine traggono le somme sul lavoro svolto fino ad ora, seguite dai futuri sviluppi ed i possibili miglioramenti applicabili al progetto.

Indice

Elenco delle figure

Capitolo 1

Descrizione del gioco

Un videogioco di strategia è per definizione un gioco in cui il risultato finale è fortemente influenzato dalle decisioni prese dal giocatore. Un esempio su tutti è rappresentato dagli scacchi.
Rapportando la stessa tipologia al mondo del mobile gaming si evince che i giochi gestionali sono i più popolari: questo genere infatti occupa diverse posizioni tra i primi posti della classifica dei giochi più installati dello store.
La struttura di base di questo tipo di giochi *freemium*[1] è comune: l'obiettivo solitamente è quello di sviluppare un villaggio, cercando di incrementare al massimo il livello degli edifici presenti, accumulando trofei e raggiungendo degli obiettivi preposti che una volta completati concedono una ricompensa in termini di risorse.
Quest'ultime possono essere investite per innalzare il livello del villaggio, cercando di potenziare il più possibile ogni costruzione e possono essere racimolate con modalità differenti: tramite strutture adibite alla generazione di ricchezze, conquiste di bottini in seguito alle battaglie o tramite delle ricompense ottenute con il raggiungimento di obiettivi.
Potenziare una struttura ha un costo in termini di risorse da pagare e tempo di attuazione dell'upgrade.

[1]Videogiochi scaricabili gratuitamente nel quale però vengono proposte funzionalità aggiuntive a pagamento, come ad esempio l'acquisto di risorse extra

Un edificio migliorato, rispetto al livello precedente, ha maggiori punti vita, miglior capacità o velocità di estrazione (nel caso di edifici per risorse), maggior danno (nel caso di edifici difensivi) ed eventualmente altre caratteristiche.
Il grado raggiunto dallo stronghold, che è l'edificio base del villaggio, denota solitamente anche il livello generale del giocatore.
Aumentando di livello di quest'ultimo sono resi disponibili nuovi edifici e nuovi livelli di potenziamento per le strutture. Il giocatore ha la possibilità di posizionare gli edifici nel punto che preferisce del villaggio, purché non si sovrappongano, e di modificarne la disposizione a piacimento.
Pertanto collocare intelligentemente gli edifici molto spesso condiziona l'esito delle battaglie in quanto una buona strategia difensiva può indurre in errore o trarre in inganno l'avversario che attacca.

Il gioco preso in considerazione è Rival Kingdoms, prodotto dalla software house Space Ape Games e scaricabile dal Google Play Store e dall'App Store. RK ha raggiunto circa 4 milioni di installazioni su Android ed è di fatto tra i più scaricati della categoria, insieme a Clash of Clans, Boom Beach e Clash of Kings.

1.1 Perché Rival Kingdoms

Interfacciarsi direttamente con il display dello smartphone introduce dei considerevoli tempi di latenza in fase di invio di segnali allo schermo. Questo ha escluso automaticamente tutti i giochi che richiedono un'interazione continua e rapida con il device, come ad esempio la maggior parte dei giochi arcade, in cui molto spesso il punteggio ottenuto è fortemente dipendente dalla reattività del giocatore.
Il focus è quindi ricaduto sui giochi strategici, dove tendenzialmente il risultato della gara viene conferito sulla base delle scelte prese dal giocatore, tollerando i tempi di latenza necessari per l'invio di input al device.

Potendo scegliere tra una vasta gamma di applicazioni disponibili, sono stati presi in considerazione i giochi online[1] più blasonati. Per il problema citato poc'anzi sono stati esclusi applicativi come *Clash Royale*[2], nel quale ci si trova a fronteggiare un avversario che risponde alle mosse in diretta.
Inizialmente è stato preso in esame *Clash of Clans*[3], che dopo qualche sperimentazione, si è dimostrato inadeguato per una serie di impedimenti prevalentemente dovuti al costo e al tempo di rigenerazione delle truppe.
Questi elementi sono però fondamentali per cercare di massimizzare il numero di battaglie e quindi raccogliere più dati possibili.
La ricerca è quindi ricaduta su Rival Kingdoms, applicativo che ricalca le meccaniche di gioco di CoC senza però imporre troppe limitazioni alle funzionalità di interesse.
Numerosi sono i punti a favore di RK rispetto a CoC, riassumibili come di seguito:

- Nessun costo di generazione per le truppe.
- Nessun tempo di latenza dovuto alla generazione delle milizie: dopo ogni battaglia queste si auto rigenerano automaticamente.
- Possibilità di combattere anche senza avere risorse.
- Fase difensiva ignorabile ai fini del gioco in quanto anche in caso di sconfitta le risorse sottratte non sono proibitive.
- Assenza di tasse da pagare per ingaggiare una battaglia.

RK introduce però due lati negativi che hanno di fatto rappresentato le due limitazioni più grandi ai fini dello sviluppo del progetto:

- Possibilità di svolgere al massimo di 72 battaglie giornaliere (1.2 per approfondimento).

[1] Giochi che permettono di combattere contro altri giocatori reali
[2] Gioco prodotto da Supercell: https://clashroyale.com
[3] https://clashofclans.com

- Algoritmo di selezione degli avversari sceglie opponenti troppo semplici da battere (circa il 90% delle battaglie vengono vinte).

Questi due punti complicano la raccolta dei dati, contribuendo inoltre a creare un dataset poco equilibrato.

1.2 Descrizione delle meccaniche di gioco

Il gioco si sviluppa essenzialmente nelle fasi di attacco e di difesa.
La fase difensiva consiste nella configurazione del proprio villaggio ma, non essendo rilevante ai fini dello scopo di questo studio, è stata totalmente ignorata.
La fase di attacco invece è la parte sulla quale è focalizzato questo lavoro e, di conseguenza, quella che verrà analizzata più approfonditamente.

1.2.1 Descrizione delle battaglie

La fase di attacco in RK non differisce di molto da tutti gli altri giochi strategici: per vincere la battaglia l'attaccante deve semplicemente abbattere lo stronghold, indipendentemente dal resto del villaggio.
Esistono diverse modalità di attacco quali battaglie multiplayer, tornei, arena, campagna o guerre tra regni, ma l'unica modalità presa in considerazione in questo studio è la battaglia multiplayer, un semplice PvP[1] online.
Nelle battaglie è importante notare come non ci sia interazione diretta e immediata tra i giocatori: chi riceve un attacco può solamente visualizzare la lotta, senza possibilità di intervenire in nessun modo.

Le pietre di battaglia, necessarie per affrontare gli avversari, vengono generate automaticamente ogni 20 minuti senza alcun costo, fino ad un massimo di cinque pietre; attaccare un avversario richiede l'impiego di una pietra per

[1] Player vs player

ogni singola battaglia. Le pietre disponibili sono visualizzabili dalla schermata del proprio villaggio insieme al countdown di generazione (fig. 1.1).

Figura 1.1: Pietre di battaglia disponibili

Una volta ingaggiata la battaglia viene caricato il villaggio dell'avversario selezionato dal sistema sullo schermo. La battaglia ha una durata massima di tre minuti e il conteggio del tempo inizia dal momento in cui viene schierata la prima truppa d'attacco.

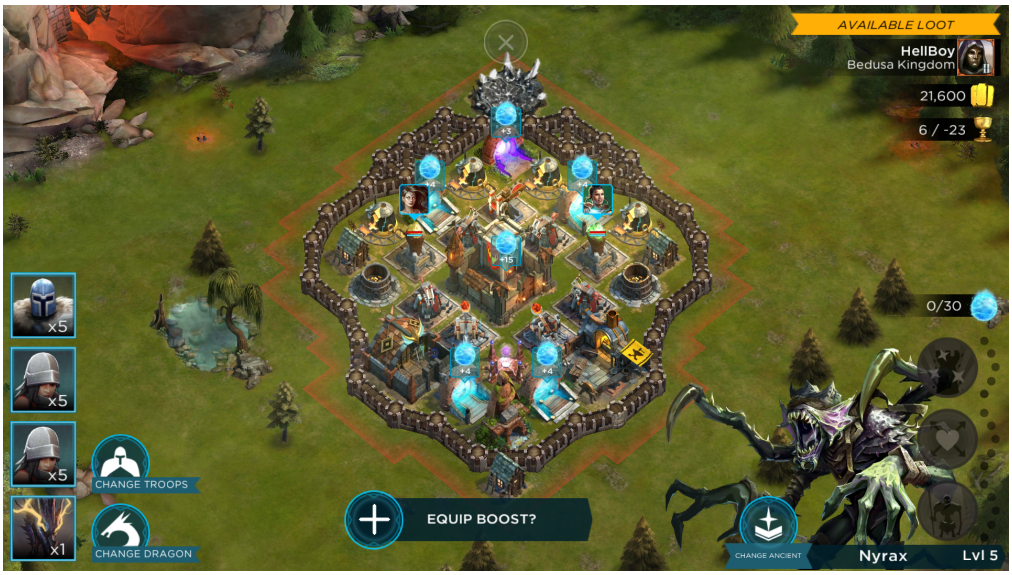


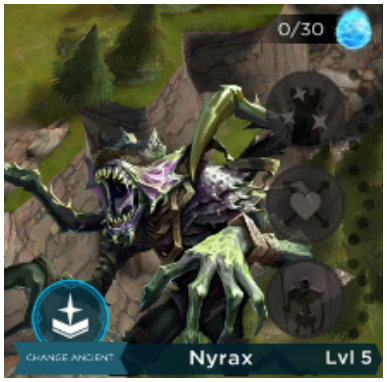
Figura 1.2: Esempio di battaglia prima di schierare truppe

Per schierare le forze armate in campo è sufficiente trascinare l'icona della truppa selezionata nella posizione scelta all'interno del villaggio.

Durante la battaglia è possibile utilizzare diverse tipologie di truppe:

- **Truppe regolari**: possono essere soldati semplici, arcieri o altre tipologie. Le truppe disponibili sono visualizzabili in basso a sinistra della schermata di battaglia (fig. 1.2 e 1.3a);
- **Drago**: utilizzabile come una truppa regolare ma dal potenziale molto più alto. Può anche sferrare attacchi aggiuntivi che però non vengono sfruttati in questo progetto (primo riquadro partendo dal basso in fig. 1.3a);
- **Ancient**: non è una vera e propria truppa ma permette di utilizzare dei poteri speciali che differiscono da ancient in ancient. Anche questa tipologia di attacco non viene utilizzata in fase di battaglia (fig. 1.3b).

(a) Esempio di milizie e drago



(b) Esempio di ancient

Figura 1.3: Esempi di truppe e ancient utilizzabili in battaglia

Le potenzialità aggiuntive offerte dal drago e dall'ancient non sono state sfruttate per un motivo ben preciso. Si può notare infatti come non sia possibile spostare le truppe una volta schierate, ad eccezione del drago. Questa possibilità è stata trascurata intenzionalmente, così come l'utilizzo dei poteri dell'ancient, per rendere l'esito della battaglia un risultato deterministico dettato solamente dalla conformazione del villaggio attaccato e dalle posizioni di schieramento delle milizie.

Per lo stesso motivo è molto importante anche sottolineare che tutte le truppe vengono piazzate il prima possibile sul terreno di gioco, proprio perché, effettuando gli schieramenti in istanti diversi, questo potrebbe risultare strategicamente rilevante ai fini del risultato finale.
Sulla base di queste semplificazioni apportate si può affermare quindi che l'esito di ogni scontro è deterministicamente decretato nel momento in cui tutte le truppe sono state collocate.
Le facilitazioni messe in atto aiutano anche a mantenere una certa semplicità di strutturazione delle informazioni relative ad una battaglia. Introdurre variabili temporali avrebbe infatti decisamente complicato la struttura dei dati descritta nella sezione 3.4.3.

Come viene spiegato nella sezione 4.3 però, le restrizioni sopra descritte non hanno comunque permesso di abbassare le percentuali di successo, che quindi rimangono un problema per quanto riguarda la fase di training della CNN[1].

1.2.2 Analisi dei risultati

Al termine di ogni scontro viene mostrata una schermata riassuntiva che ricapitola il risultato della battaglia (vittoria o sconfitta) ed i premi ottenuti. In figura 1.4 è visualizzabile uno screenshot di esempio.
Per acquisire la percentuale di distruzione di ogni battaglia invece è necessario spostarsi nella sezione *battle history*, accessibile dal menù di gioco nella home. Le informazioni dello scontro verranno riportate come l'esempio in figura 1.5.

[1]Convolutional Neural Network

Figura 1.4: Esempio di schermata riassuntiva alla fine del combattimento

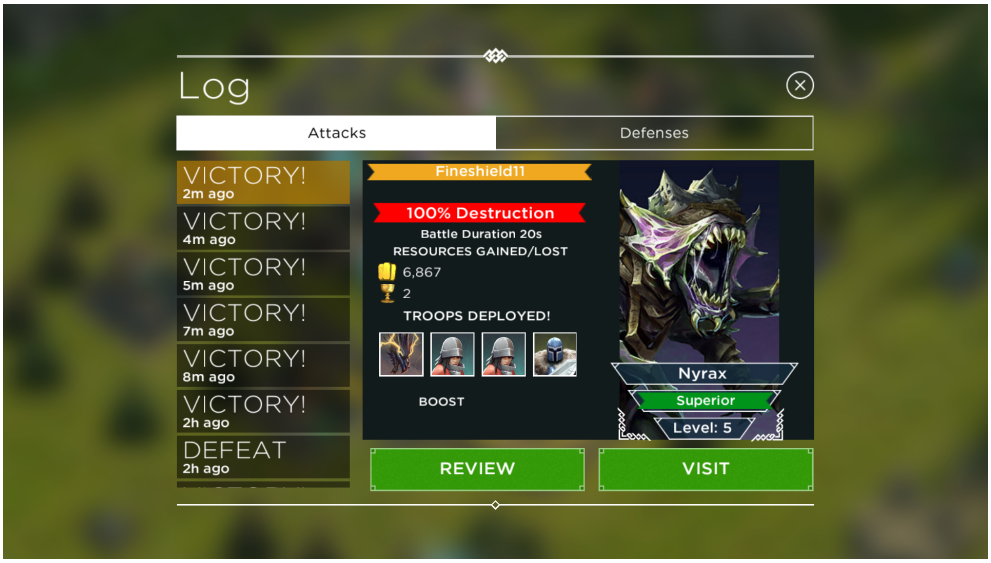


Figura 1.5: Esempio di statistiche nella Battle History

Capitolo 2

Controllo del device

Un punto di forza dell'applicativo sviluppato è la sua potenziale adattabilità alla maggior parte dei dispositivi che siano dotati di un sistema operativo Android.
Inoltre, grazie all'utilizzo dei comandi di simulazione delle gesture di ADB è possibile simulare a tutti gli effetti un reale utente.

D'altra parte l'utilizzo di questa tecnologia di interfacciamento con i device ha compromesso la scelta del gioco sul quale sviluppare il progetto.
Per le gesture più semplici come il tap e lo swipe il tempo che intercorre tra il lancio del comando e l'effettivo verificarsi dello stesso è di circa un secondo[2].
Se invece si parla di gesture più complicate, come il pinch out, che richiede di essere frammentata in tanti piccoli eventi sullo schermo, i tempi si dilatano considerevolmente per arrivare fino a venti secondi di durata effettiva.
Per ovviare a tale limitazione sono stati presi in considerazione soltanto giochi in cui l'efficacia dei comandi impartiti non sia dipendente dalle tempistiche con il quale essi vengono lanciati.
A tale proposito i giochi strategici/gestionali, con meccaniche di gioco descritte nel capitolo precedente, sono stati individuati come i più indicati.

[2]I tempi sono stati registrati con una connessione ADB via cavo e con emulatore

Di seguito vengono descritte le modalità con cui avviene l'interfacciamento del bot sviluppato con il device, analizzando il mezzo che permette la comunicazione e le procedure implementate per sfruttarne al meglio le potenzialità.

2.1 Device supportati

Nonostante la vastità dei dispositivi che utilizzano un sistema operativo basato su Android, l'applicativo sviluppato è in grado di girare sulla gran parte di essi, compresi gli emulatori. Gli unici requisiti richiesti per l'utilizzo dell'applicativo sono infatti:

- Disponibilità di connessione del device ad una macchina (sia via cavo che tramite una connessione wifi) per permettere l'utilizzo di una adb shell;
- Aspect ratio dello schermo del device di rapporto 16:9 (la maggior parte dei dispositivi in circolazione in questo momento hanno questo rapporto).

Durante lo sviluppo sono stati utilizzati prevalentemente due dispositivi: un Nexus 5 (sia fisico che emulato) ed un Nexus 5X emulato. I dispositivi hanno entrambi uno schermo con risoluzione in fullHD[1] con rapporto 16:9.

2.2 Android Debug Bridge

ADB shell è un tool da riga di comando che permette di comunicare con i dispositivi Android, che siano essi emulati (macchine virtuali) o fisici.
Il programma è del tipo client server e include tre componenti:

- Client: viene eseguito sulla macchina utilizzata per lo sviluppo. Si può richiamare un client da terminale oppure utilizzando tool dedicati sviluppati da terze parti;

[1] 1920x1080 pixel

- Server: viene eseguito come processo di background sull'emulatore o sul device usato per i test;

- Demone: viene eseguito come un processo in background su ogni istanza dell'emulatore o del dispositivo.

ADB è incluso nel pacchetto Android SDK Platform-Tools[1], contenuto a sua volta nell'SDK Manager. Infatti il pacchetto viene installato in *android_sdk/platform-tools/*.

All'avvio, un client ADB controlla in primo luogo se vi è un processo server adb già in esecuzione. Se non è presente, il task viene avviato.
Quando il server si avvia stabilisce una connessione TCP sulla porta locale 5037 e si mette in attesa di ricevere comandi inviati dai client ADB.
Questi ultimi infatti utilizzano proprio la porta 5037 per tentare di stabilire la connessione.
Il server apre quindi una comunicazione TCP privata per ogni device che risulta disponibile ed in esecuzione. L'intervallo di porte dispari che vanno dalla 5555 alla 5587 sono infatti dedicate agli emulatori e ai dispositivi. Si noti che ogni client utilizza due porte in sequenza: la porta pari per la console e quella dispari per la connessione adb. Quindi in un esempio con due dispositivi connessi avremo la seguente situazione:

```
Emulator 1, console: 5554
Emulator 1, adb: 5555
Emulator 2, console: 5556
Emulator 2, adb: 5557
```

Come mostrato, l'emulatore connesso alla porta 5555 è lo stesso la cui console è in ascolto sulla porta 5554.
Una volta che le connessioni sono state stabilite è possibile utilizzare i comandi ADB per accedere a tali device.

[1]https://developer.android.com/studio/releases/platform-tools.html

Nel nostro caso di studio la shell in questione è stata utilizzata come client al fine di impartire comandi al dispositivo collegato.
Nelle prossime sezioni vengono descritte le primitive che sono state adoperate per l'interfacciamento al dispositivo.

2.2.1 Screenshot

L'acquisizione dello screenshot è una delle più importanti funzioni sul quale si basa il funzionamento di tutto l'applicativo.
Combinata con la *ricerca parziale d'immagine* (sez. 2.4.1) e con la funzione di *cosine similarity* (sez. 2.4.2) questa primitiva permette allo script di muoversi e localizzarsi all'interno dell'interfaccia del gioco, consentendo all'applicativo di individuare ad esempio le corrette coordinate di un bottone da premere o una porzione di schermo che contiene informazioni rilevanti.
Sfortunatamente non è possibile ottenere il file immagine generato direttamente sulla macchina che esegue lo script senza passare per il dispositivo stesso. Sarà quindi necessario l'uso combinato di più comandi per ottenere l'immagine pronta ad essere analizzata dall'applicativo.
Questi sono gli step da seguire per arrivare al risultato ricercato:

1. Generazione dell'immagine in memoria del device tramite la funzione ADB di *screencap*:

   ```
   $> adb shell screencap -p DEVICE_FILE_PATH
   ```

   L'immagine viene salvata in full size, a seconda della risoluzione dello schermo del device che si sta utilizzando.

2. Invio di una copia del file appena generato alla macchina che esegue lo script:

   ```
   $> adb pull DEVICE_FILE_PATH (COMPUTER_FILE_PATH)
   ```

   Se non viene specificato il path locale (della macchina che esegue lo script) viene utilizzata la posizione attuale.

3. Rimozione del file originale in memoria del device per non occupare inutilmente memoria:

```
$> adb shell rm DEVICE_FILE_PATH
```

4. Resize dell'immagine ad una risoluzione più bassa utilizzando la libreria OpenCV[2]:

```
img = cv2.imread(nameFile) # apertura dell'immagine
img = cv2.resize(img, (0,0), fx=RES_SCALE,
    fy=RES_SCALE)
cv2.imwrite(nameFile ,img) # scrittura dell'immagine
```

   RES_SCALE è la costante che definisce il fattore di abbassamento di qualità di immagine.
   Dato che la risoluzione originale degli screenshot è risultata troppo alta per lo scopo si è deciso di lavorare su immagini di risoluzione 768x432.
   Nel caso di schermi fullHD di risoluzione 1920x1080 la costante RES_SCALE dovrà essere impostata a 0.4, per ottenere un'immagine con risoluzione al 40% rispetto all'originale.
   Da questo possiamo quindi appurare che l'applicativo sviluppato è compatibile con tutti i device che hanno uno schermo in 16:9.
   Sarà sufficiente aggiornare il valore di RES_SCALE per ottenere immagini di dimensione 768x432.
   Per esempio, se disponiamo di un device con risoluzione HD (1280x720) il fattore di abbassamento di risoluzione andrà impostato a 0.6 (60% dell'immagine originale) per raggiungere le dimensioni compatibili all'applicativo.

2.2.2 Tap

ADB fornisce delle api di alto livello per emulare un semplice tap sullo schermo. Il comando da utilizzare prende per parametri le coordinate (X, Y) del punto in cui si vuole riprodurre il tocco.

```
$> adb shell input tap X Y
```

Da notare che, lavorando con immagini di risoluzione 768x432, le coordinate (X, Y) sulle quali inviare il tap vanno riadattate a seconda della risoluzione dello schermo del device che si sta utilizzando, ovvero riapplicando all'inverso il fattore di abbassamento di risoluzione (RES_SCALE, introdotto alla sezione 2.2.1) per ottenere delle coordinate consone al display utilizzato.
Ad esempio, se l'applicativo ha necessità di emulare un tap nelle coordinate (200, 300), il comando adeguato da lanciare sarà il seguente:

```
$> adb shell input tap 500 750
```

Supponendo di utilizzare uno schermo FullHD infatti avremo un RES_SCALE pari a 0.4. Per adattare le coordinate dalla risoluzione che utilizza l'applicativo alla risoluzione reale dello schermo sarà necessario dividere 200 e 300 per RES_SCALE.

2.2.3 Swipe

Come per il tap, ADB fornisce anche un comando per riprodurre una gesture di swipe sullo schermo.

```
$> adb shell input swipe X1 Y1 X2 Y2 T
```

In questo caso è necessario specificare le coordinate dei punti di partenza e di arrivo della gesture ed in ultimo il tempo (espresso in millisecondi) con il quale la gesture deve concludersi, ovvero quanto velocemente il tocco simulato si sposterà dal primo al secondo punto.

```
$> adb shell input swipe 246 328 786 675 1000
```

L'esempio riportato lancia uno swipe dal punto (246, 328) al punto (786, 675) in un secondo.

Come per il tap, anche le coordinate dei punti specificate per lo swipe devono essere adattate a seconda della risoluzione del device dividendo i punti per il fattore di riduzione di qualità RES_SCALE.

2.2.4 Pinch out

Differentemente dal tap e dallo swipe, lanciare una gesture di pinch out (o zoom out) non è del tutto triviale. La particolare gesture in questione utilizza multitouch, ovvero più tocchi simultanei sullo schermo. In particolare, un pinch out utilizza due swipe contemporanei che convergono in un punto centrale, come illustrato in figura 2.1.

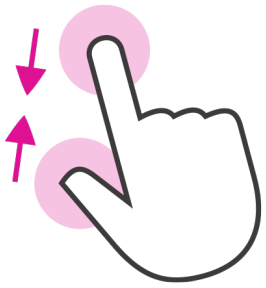


Figura 2.1: Pinch out gesture

Per emulare il pinch out è necessario riprodurre fisicamente una volta la gesture sul dispositivo, registrando ogni singolo evento generato.
Una volta memorizzati tali eventi sarà necessario generarli in sequenza per riprodurre il movimento desiderato all'occorrenza.

ADB fornisce le funzioni *getevent* e *sendevent*, rispettivamente per ottenere e riprodurre gli eventi di touch. Di seguito le modalità con le quali viene impiegata la primitiva *getevent* per registrare la gesture:

```
$> adb shell getevent | grep dev/input/event1 >
    getevent_input.txt
```

Questo comando avvia un processo che fino a quando non viene stoppato rimane in ascolto degli eventi generati dal touchscreen, scrivendoli su un file di testo.
Il parametro *dev/input/event1* identifica l'input device da registrare (in questo caso è il touchscreen).
Solitamente il pannello touch è quasi sempre identificato con *event1* in quanto si presume essere il dispositivo di input principale.
Risulta comunque possibile verificare il nome del dispositivo interessato lanciando soltanto il comando primitivo:

```
$> adb shell getevent
```

Dopo aver avviato il processo basterà riprodurre un qualsiasi tocco sullo schermo per vedere un risultato simile al seguente sul terminale (stdout):

```
add device 1: /dev/input/event0
  name:     "Power Button"
add device 2: /dev/input/event1
  name:     "touchscreen"
/dev/input/event1: 0003 0039 00000000
/dev/input/event1: 0003 0030 000001d0
/dev/input/event1: 0003 003a 00000081
...
```

Come prima cosa il comando stampa tutti i device di input che rileva. In questo caso è stato identificato il device 1 (*/dev/input/event0*) come tasto di accensione e device 2 (*/dev/input/event1*) come pannello di touchscreen.

Dopo aver campionato la gesture tramite la funzione *getevent*, è necessario adattare i risultati ottenuti per essere poi correttamente utilizzati dalla funzione *sendevent*.
La struttura dei parametri rimane la stessa con l'unica differenza che la funzione *getevent* ritorna valori numerici in esadecimale, mentre la *sendevent* va utilizzata con numeri in base 10. Pertanto risulta necessario operare una conversione di base.
La chiamata di funzione richiede la seguente struttura:

```
sendevent [device] [event_type] [event_code] [value]
```

Ci sono soltanto due tipi di evento utilizzabili:

- **EV_ABS (3):** identifica un evento assoluto e viene utilizzato per impostare i vari parametri;
- **EV_SYN (0):** descrive un evento di sincronizzazione e viene utilizzato per rendere effettive le modifiche descritte precedentemente tramite eventi assoluti.

Sono invece diversi i codici di evento utilizzabili. Se il nome riporta come prefisso *ABS* allora l'evento dovrà essere di tipo assoluto; allo stesso modo, se preceduto da *SYN*, l'evento serve per la sincronizzazione. Di seguito vengono riportati i più importanti.

- **ABS_MT_TRACKING_ID (57):** codice identificativo del touch (molto importante per eventi di multitouch);
- **ABS_MT_POSITION_X (53):** set della coordinata x del tocco;
- **ABS_MT_POSITION_Y (54):** set della coordinata y del tocco;
- **ABS_MT_TOUCH_MAJOR (48):** larghezza del dito che toccherà lo schermo espresso in pixel;
- **ABS_MT_PRESSURE (58):** pressione dello schermo;

- **SYN_MT_REPORT (2):** chiusura di un evento di touch (sollevamento del dito);
- **SYN_REPORT (0):** commit degli absolute events specificati precedentemente e chiusura della comunicazione.

Di seguito un esempio di evento dato in output dalla funzione *getevent* poi adattato per una chiamata *sendevent*:

```
getevent /dev/input/event1 0003 0036 1647
sendevent /dev/input/event1 3 54 5703
```

Qui viene impostata la coordinata y a 5703px per un ipotetico tap o swipe sullo schermo.

2.3 ADB tramite Python

Per utilizzare i suddetti comandi ADB è necessario essere in grado di lanciare nuovi processi tramite Python, il linguaggio nel quale è scritto il bot. Con il modulo *subprocess*[1] è possibile creare nuovi processi con la possibilità di controllare anche le relative pipe.
In particolare in questo progetto è stato utilizzata la funzione *check_output* che prende per argomento una lista contenente il comando da lanciare, opportunamente diviso ad ogni spazio, e ritorna una stringa di byte che rappresenta l'output ottenuto.
Di seguito un esempio che lancia un tap sullo schermo:

```
sub.check_output(["adb", "shell", "input", "tap", "200", "300"])
```

[1] https://docs.python.org/3/library/subprocess.html

Qui invece viene riportato lo snippet di codice implementato nel progetto che definisce la funzione utilizzata per lanciare ogni comando adb:

```
def launch_cmd(cmd):
    output = sub.check_output(cmd.split())
    return output.decode('UTF-8', 'ignore').split()
```

Il valore di ritorno è la decodifica in UTF-8 dell'output generato.

2.4 Muoversi all'interno dell'app

Per poter sfruttare le funzionalità del gioco il bot deve essere in grado di transitare attraverso le diverse view dell'applicazione.
Le schermate principali da navigare sono:

- **Home:** viene mostrata la conformazione del proprio villaggio e tutte le informazioni relative ad esso. Questa schermata è il punto di partenza per accedere a tutte le altre (fig. 2.2a);
- **Mappa:** da qui è possibile selezionare le varie modalità di battaglia e iniziare a combattere (fig. 2.2b);
- **Schermata di battaglia:** campo di gioco dove sfidare gli avversari(fig. 1.2);
- **Menù:** accessibile soltanto passando dalla home, è l'unico modo per accedere alla *battle history* (fig. 2.2c);
- **Battle history:** storico delle battaglie con il relativo esito. Viene utilizzato per acquisire la percentuale di distruzione di ogni attacco effettuato (fig. 1.5).

Nelle prossime sezioni vengono illustrati i due differenti metodi implementati per raggiungere lo scopo.

2.4.1 Template matching

Il *Template matching* è un metodo per ricercare e trovare la posizione di un template all'interno di un'immagine più grande.
La libreria OpenCV[2] fornisce la funzione *matchTemplate*[1] per lo scopo.
Il funzionamento è intuitivo: la figura di template viene fatta scorrere sopra l'immagine in input (operando una sorta di convoluzione in 2D) e viene confrontata con la porzione di immagine di input sottostante.
OpenCV mette a disposizione diversi metodi di confronto. In questo progetto è stato utilizzato il TM_CCORR_NORMED, che applica la seguente formula ad ogni comparazione:

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

dove
T(x,y) = valore immagine di template in posizione (x,y)
I(x,y) = valore immagine di input in posizione (x,y)

La funzione ritorna un'immagine in scala di grigi dove ogni pixel denota la vicinanza del pixel stesso al corrispondente del template.
Una volta ottenuto l'output, è possibile utilizzare la funzione *minMaxLoc* per localizzare le corrispondenze riscontrate.
Ad ogni massimo rilevato corrisponderà l'angolo in alto a sinistra della porzione di immagine (con dimensioni uguali a quelle del template) che racchiude un match. In fase di sviluppo sono stati acquisiti diversi template, ritagliati delle view precedentemente elencate (fig. 2.4). Questi ultimi vengono impiegati con il *template matching* per due scopi:

- Riconoscimento della view attualmente visualizzata in primo piano: si ricerca un template peculiare che sia presente soltanto in una determi-

[1] http://bit.ly/2taGoBf

nata schermata (ad esempio il tasto *Map* in figura 2.4 sarà riconoscibile soltanto dalla home, come evidenziato in figura 2.3);

- Localizzazione del punto del display sul quale effettuare una gesture: ad esempio, ammettendo di trovarci nel menù in figura 2.2c e di voler tornare alla home, sarà necessario ricercare la posizione del bottone *close* (ultimo in fig. 2.4) per poter poi inviare un tap su di esso.

2.4.2 Cosine similarity

La similarità del coseno (o cosine similarity) è una tecnica utilizzata per misurare la vicinanza tra due vettori. Viene impiegata in diversi ambiti quali l'image processing o la segmentazione di immagine[1]. Il calcolo di tale misura è basato sul coseno della differenza tra gli angoli dei due elementi in input. Infatti due array che hanno un angolo con la stessa inclinazione restituiranno valore 1. Al contrario, due vettori diametralmente opposti (con distanza di 180°) restituiranno valore -1. Se invece la differenza di inclinazione è di 90° avremo valore 0 in output.
Il coseno di similitudine tra due vettori A e B è quindi definito come segue:

$$CS(x, y) = \frac{A \cdot B}{||A|| \cdot ||B||} = \frac{\sum_{i=1}^{n} A_i \cdot B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Questa tecnica viene comunemente utilizzata negli ambiti sopracitati soprattutto per la sua facilità di utilizzo, dovuta al fatto di ritornare sempre valori compresi tra -1 e 1.

Il calcolo del cosine similarity viene effettuato utilizzando la funzione *cosine* dalla libreria *scipy*[3] e viene impiegato al fine di riconoscere la view in primo piano sul display.
Oltre ai template descritti nella sezione precedente, sono stati quindi acquisiti anche degli screenshot generici di ogni view al fine di essere confrontati con la schermata in foreground sul display.

[1] Un esempio pratico può essere il riconoscimento del volto[16]

La funzione *cosine* calcola il valore di CS tra i due vettori unidimensionali presi in input.
Per poterla utilizzare è stato infatti necessario operare un appiattimento (*flattening*) delle immagini per portarle ad una sola dimensione. Per questo compito è stata impiegata la libreria *numpy*[4], in particolare la funzione *ravel* [1].

Di seguito il codice impiegato per operare il confronto:

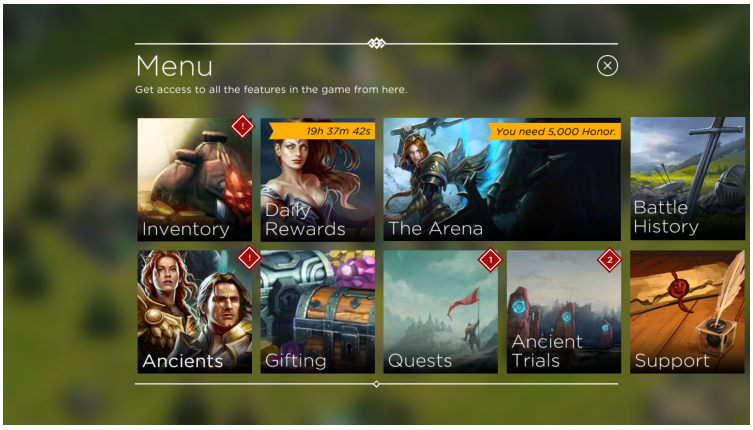
```
def cosine_sim(s_name):
    sample = np.ravel(cv2.imread(s_name).astype(float))
    screen = take_screenshot()
    to_test = np.ravel(cv2.imread(screen).astype(float))
    cs = cosine(sample, to_test)
    remove(screen)
    print ("Cosine similarity: ", cs)
    if cs < 0.08:
        return True
    else:
        return False
```

La funzione definita prende in input un'immagine di template (*s_name*) e lo confronta con lo screenshot appena acquisito.
Nella comparazione è stata impostata una soglia dello 0.08: se l'indice di similitudine restituito è inferiore viene ritornato il valore True, ad indicare che attualmente in foreground sullo schermo è presente la view del template, altrimenti viene restituito False per denotare una sostanziale differenza di schermata.
Il valore di soglia è stato decretato in seguito alle verifiche fatte sulla base delle possibili minime differenze tra due istanze della stessa view.

[1] https://docs.scipy.org/doc/numpy/reference/generated/numpy.ravel.html

(a) Home

(b) Mappa



(c) Menù

Figura 2.2: Schermate principali del gioco

Figura 2.3: Match del template all'interno del display

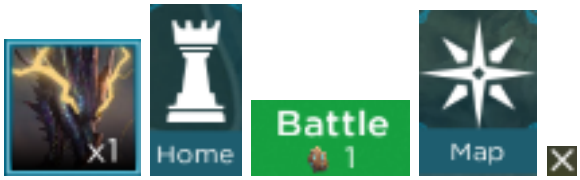


Figura 2.4: Alcuni template di esempio

Capitolo 3

Svolgimento delle battaglie

In questo capitolo viene descritta la struttura di ogni iterazione che compie il bot per portare a termine una battaglia, partendo dalla visualizzazione del villaggio per arrivare fino al salvataggio dei dati ottenuti.

Lo scopo principale è quello di riuscire ad ottenere dei dati informativi e strutturati sull'avversario fronteggiato e sulle mosse compiute dal bot durante la battaglia. Al termine di ogni lotta il bot deve essere in grado di produrre in output un record che contenga le seguenti informazioni:

- screenshot del villaggio;
- posizionamento del villaggio in relazione a tutto il campo di gioco;
- posizione dello stronghold;
- livello generale del giocatore;
- conformazione dell'esercito impiegato in battaglia;
- dislocamento delle truppe;
- risultato ottenuto dalla battaglia.

Nei prossimi paragrafi verranno ripercorsi tutti i passaggi compiuti dal bot al fine di ottenere il risultato ricercato.

3.1 Acquisizione dell'immagine

Il primo passo fondamentale per iniziare una battaglia è riuscire ad acquisire un'immagine il più descrittiva possibile del terreno di gioco. Una volta che l'algoritmo di matchmaking ha selezionato l'avversario contro cui lottare, viene predisposto a schermo il villaggio da distruggere.
Al completamento del caricamento sarà possibile studiare il campo di gioco, operando gesture di zoom e swipe, oppure toccando ogni singolo elemento per leggere livello e tipo di edificio selezionato.
Lo scopo in questa fase è quello di ottenere un'immagine che raffiguri tutto il campo di gioco e che mantenga un posizionamento standard di quest'ultimo, al fine di poter localizzare più facilmente gli elementi contenuti in esso.

Inizialmente il campo di gioco non viene visualizzato nella sua interezza: al centro dello schermo viene sempre posto il municipio, con uno zoom che varia a seconda dell'estensione delle strutture posizionate nel suo intorno. Più il villaggio sarà esteso e minore sarà lo zoom applicato; lo scopo di questo posizionamento è quello di mettere in foreground l'obiettivo principale, che è lo stronghold, e di adattare poi il resto della visualizzazione, per cercare di mostrare quante più strutture possibili.
Questo posizionamento dinamico operato di default dal sistema complica però il processo di riconoscimento del terreno di gioco, in quanto non sono presenti dei punti fissi che possono essere identificabili come punti di riferimento: di conseguenza risulta difficile trovare una metrica efficiente per calcolare posizionamenti assoluti all'interno del campo.
Dunque è necessario adoperare un riposizionamento strategico della view prima di poter compiere qualsiasi altra azione. La configurazione ricercata deve permettere la visualizzazione dell'intero campo di battaglia e allo stesso tempo essere replicabile per tutte le possibili disposizioni che può assumere un villaggio.
La soluzione ideata consiste nel visualizzare quanto più campo possibile operando uno zoom all'indietro e poi, con uno swipe, spostare la visualizzazione

in un punto comune che permetta di visualizzare al meglio il terreno.

Figura 3.1: Gesture di posizionamento

Come illustrato in figura 3.1 vengono quindi inviate due differenti gesture:

1. Pinch out (2.2.4);
2. Swipe (2.2.3) dall'angolo in basso a destra a quello opposto.

Gli effetti di queste variazioni di schermata permettono di visualizzare il campo di gioco nella sua interezza e, soprattutto, di poter identificare univocamente un determinato punto del campo esprimendolo in termini di coordinate in pixel all'interno dell'immagine.
In figura 3.2 si possono notare gli effetti del posizionamento eseguito, notando come la zona evidenziata in rosso, raffigurante l'intero terreno di gioco, rimanga la stessa per qualsiasi battaglia. Una volta che l'immagine è stata riposizionata è possibile catturare lo screenshot.
Le dimensioni dell'immagine inoltre non possono essere mantenute alla risoluzione originale: una qualità troppo alta potrebbe non aggiungere informazioni rilevanti, impiegando inutilmente troppe risorse durante la fase di training. Per i dettagli del resize si rimanda alla sezione 2.2.1.

Figura 3.2: Evidenziazione del terreno di gioco dopo il riposizionamento della view

3.2 Image processing

In seguito all'acquisizione dello screenshot, si passa alla fase di analisi dello stesso.
Le informazioni che devono essere estrapolate dall'immagine ottenuta sono essenzialmente due: il dislocamento degli edifici del villaggio, necessario per comprendere il quali posizioni sarà possibile piazzare le milizie, e la posizione del municipio, essenziale per focalizzare l'attacco direttamente sul punto più nevralgico del villaggio.

3.2.1 Localizzazione dello stronghold

Come anticipato precedentemente riuscire ad abbattere il municipio è l'unico modo per vincere la battaglia, di conseguenza conoscere la sua posizione permette di effettuare un attacco più mirato ed efficace.
Per trovare la posizione dello stronghold la tecnica utilizzata è la ricerca di un template all'interno di un'immagine, descritta nella sezione 2.4.1.

Sono stati quindi campionati diversi template, ritagliando dagli screenshot dei villaggi diversi municipi di ogni livello.
Questi template vengono poi utilizzati, partendo da quello di livello uno e proseguendo fino al raggiungimento di una corrispondenza, per eseguire una ricerche in serie. Una volta trovato un match automaticamente si conoscerà sia la posizione che il livello del municipio.

Questa metodologia introduce però una problematica: gli edifici limitrofi allo stronghold spesso si disegnano sopra di esso, complicando quindi il riconoscimento dei template.
Per ovviare al problema si è cercato di identificare precisamente la porzione di edificio che non viene mai sovrapposta da altre strutture, riducendo di fatto la dimensione dei template. Per non incorrere in falsi positivi dovuti alle dimensioni delle immagini troppo ridotte si è inoltre deciso di applicare un threshold leggermente più alto di quello utilizzato normalmente, al fine di rendere la ricerca più puntuale.

3.2.2 Ricerca del villaggio

Conoscere dove sono dislocate tutte le strutture è essenziale per la fase di schieramento in battaglia delle truppe; non si possono infatti posizionare combattenti sopra le costruzioni, bensì soltanto sulle porzioni di terreno dove non sono presenti edifici.
Per riuscire nell'intento, la prima operazione da compiere è una sottrazione di immagini.
I due operatori della sottrazione sono lo screenshot del campo e un'immagine di background (fig. 3.3), ovvero una qualsiasi immagine di un villaggio al quale però sono stati cancellati tutti gli edifici presenti nella regione di gioco, lasciando sostanzialmente soltanto il prato raffigurato. Per operare la sottrazione viene impiegata la funzione *subtract* della libreria OpenCV, che restituisce un risultato come a quello rappresentato in figura 3.4. Il risultato della sottrazione viene poi convertito in scala di grigi, in quanto i colori non

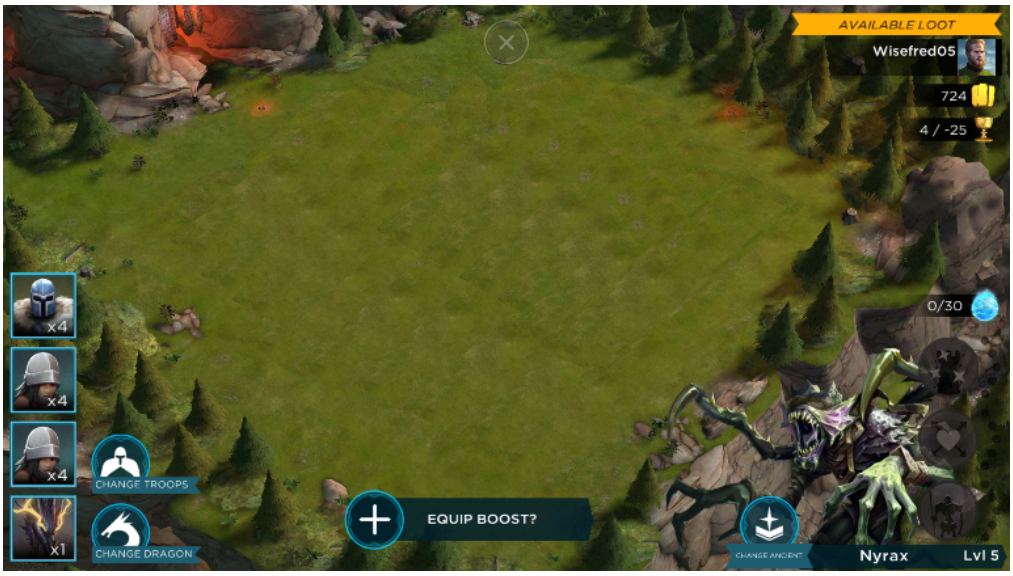


Figura 3.3: Background utilizzato nella sottrazione

contengono informazioni rilevanti allo scopo; inoltre tale operazione agevola l'applicazione di un threshold binario[1], un'operazione semplice ed efficace che permette di accentuare le differenze riscontrate in fase di sottrazione. Il valore del threshold utilizzato rappresenta una sorta di soglia di sbarramento con il quale vengono confrontati tutti i pixel dell'immagine in input: se il singolo pixel è più grande della soglia allora gli viene assegnato colore bianco, altrimenti nero.
La risultante di queste operazioni è un'immagine simile a quella riportata in figura 3.5.

Per accentuare ancora di più le differenze riscontrate nella fase precedente viene eseguita una dilatazione con la funzione *dilate*[2], in base al quale ogni pixel dell'immagine presa in input assume il valore più alto tra i pixel nel suo intorno. Si può esprimere tale operazione attraverso la seguente formula:

$$dst(x, y) = \max_{(x',y'):kernel(x',y')\neq\emptyset} src(x + x', y + y')$$

[1]https://docs.opencv.org/trunk/d7/d4d/tutorial_py_thresholding.html
[2]http://bit.ly/2CaISDa

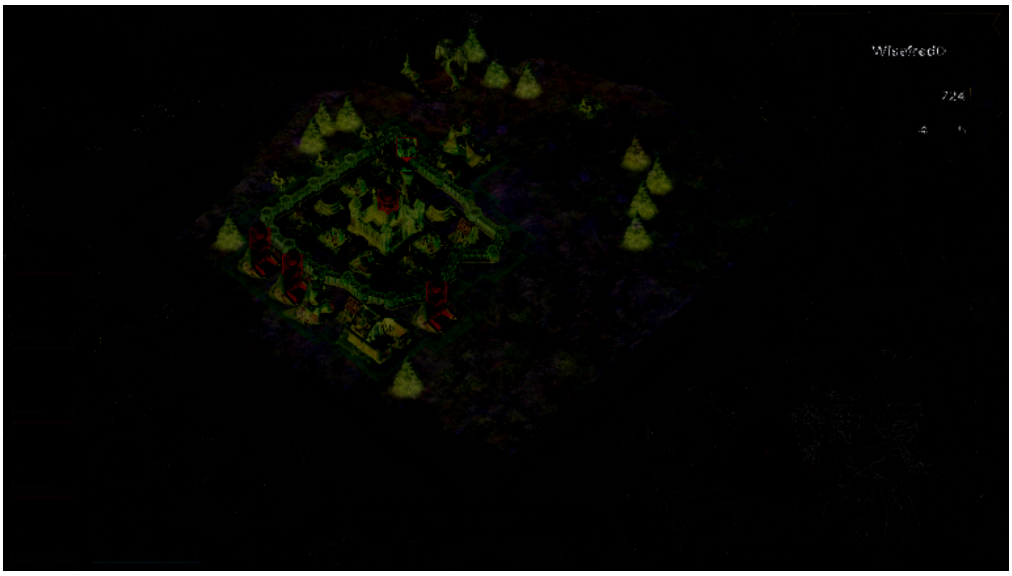


Figura 3.4: Risultato della sottrazione tra background e villaggio

dove per *kernel* si intende la forma e la dimensione dei pixel da prendere in considerazione in ogni confronto eseguito.
La dilatazione può essere applicata più volte per accentuare ancora di più le differenze. In questo caso si è deciso di compiere 8 iterazioni.

Di seguito lo snippet di codice utilizzato per arrivare al risultato mostrato in figura 3.6:

```
diff = cv2.subtract(background, camp)
diff_gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
ret, diff_gray = cv2.threshold(diff_gray, 47, 255,
    cv2.THRESH_BINARY)
kernel = np.ones((3,3),np.uint8)
dilated = cv2.dilate(diff_gray, kernel, iterations=8)
```

Una volta messe in evidenza le differenze, l'ultima operazione da compiere è definire con precisione i contorni attraverso l'utilizzo della funzione *findCon-*

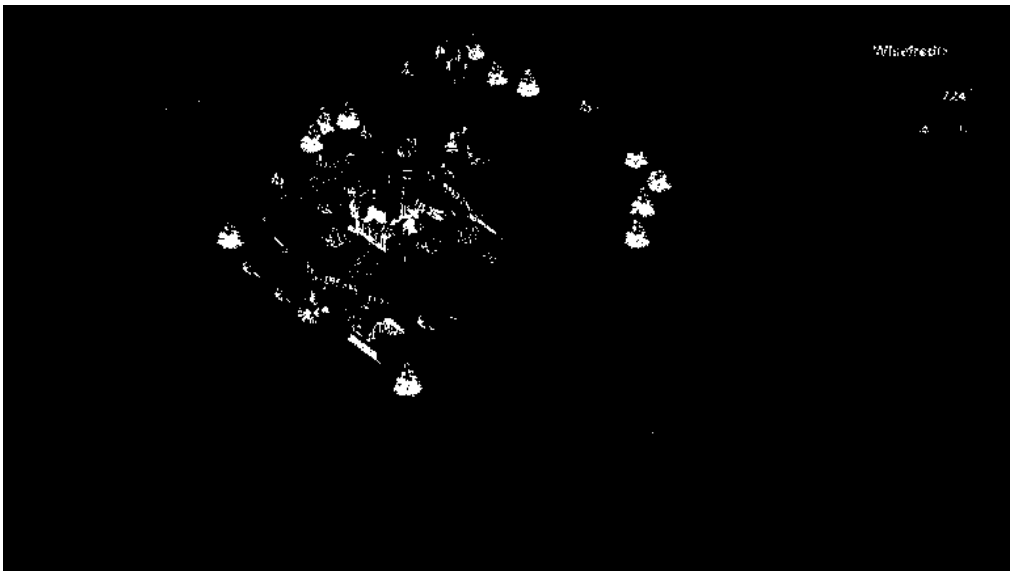


Figura 3.5: Conversione in scala di grigi e applicazione del threshold al risultato della sottrazione

tours[1] di OpenCV, che prende in input l'immagine dilatata e restituisce un set di contorni rilevati. Qui riportata la chiamata a funzione effettuata:

```
with_cont, contours, _ =
  cv2.findContours(dilated,cv2.RETR_EXTERNAL,
  cv2.CHAIN_APPROX_SIMPLE)
```

dove i 3 parametri rappresentano rispettivamente:

- L'immagine binaria da prendere in esame (*dilated*, già in formato binario in seguito all'applicazione del threshold).

- La modalità di rilevazione dei contorni: in questo caso è stata utilizzata una metodologia che restituisce soltanto i contorni più esterni; ciò significa che se c'è un contorno strettamente contenuto in un altro soltanto il più esterno viene considerato.

[1] http://bit.ly/2Cbp6aD

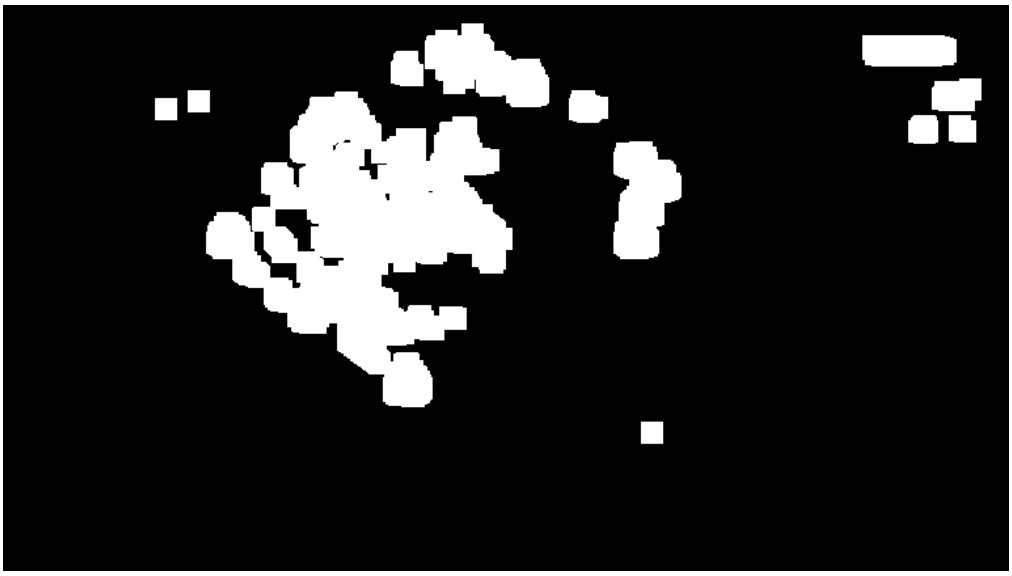


Figura 3.6: Dilatazione delle differenze riscontrate sul campo

- Il metodo di approssimazione dei contorni: serve per risparmiare spazio di memorizzazione. Ad esempio, nel caso in cui si abbia una linea retta riscontrata come contorno e si utilizzi un'approssimazione semplice, vengono memorizzati soltanto i due estremi piuttosto che tutti i punti contenuti in essa.

3.2.3 Associazione tra stronghold e contorni

Una volta esaminati i contorni e localizzato il municipio, è possibile incrociare tali dati per individuare la zona nevralgica dell'avversario, in modo da poter indirizzare direttamente le truppe intorno quell'area.
Solitamente per avere una difesa efficace si tenta di proteggere lo stronghold, piazzando nel suo intorno quante più strutture possibili per rallentare l'avanzata delle truppe offensive; tendenzialmente quindi la zona in cui sono concentrate più strutture è anche quella in cui vi sarà contenuto il municipio. A tal proposito, viene controllata la presenza dello stronghold nei contorni rilevati precedentemente, ordinati per estensione decrescente.
La funzione utilizzata per verificare se un punto è contenuto in un preciso

contorno è *pointPolygonTest*[1], che prende in input un contorno, un punto da testare e un valore booleano *measureDist*; restituisce valori positivi se il punto è dentro il contorno, negativi se è fuori o zero se il punto risiede precisamente sul limite.
Se *measureDist* è impostato a *True* i valori restituiti rappresentano la distanza che intercorre tra il punto e la linea di contorno, altrimenti vengono restituiti valori standard quali 1, -1 e 0.
Di seguito il frammento di codice che verifica tutti i contorni:

```
while cv2.pointPolygonTest(cont, (stronghold_x,stronghold_y),
    False) < 0 :
    zipped.remove(max(zipped, key=lambda item:item[0]))
    cont = max(zipped, key=lambda item:item[0])[1]
```

All'uscita dal ciclo while, nella variabile *cont* vi sarà la zona di campo ricercata, che verrà quindi selezionata come il target da distruggere.
Identificato il contorno di interesse l'ultimo passo da compiere in questa fase è quello di disegnare un'ellisse che tenti di circoscrivere il contorno, allo scopo di definire dei possibili punti sui quali schierare le truppe.
La funzione *fitEllipse*[2] di OpenCV prende come parametro il contorno e restituisce un ellisse, espresso con tre parametri:

- center: punto che identifica il centro dell'ellisse;
- axes: lunghezza del semiasse orizzontale (x) e verticale (y) dell'ellisse;
- angle: inclinazione dell'ellisse in gradi.

Di seguito l'estratto di codice per l'elaborazione dell'ellisse:

```
ellipse = cv2.fitEllipse(contour)
(x_center, y_center), (x_ax, y_ax), angle = ellipse
```

[1] http://bit.ly/2osXqFd
[2] http://bit.ly/2oqjG2f

Tutto il processo di analisi termina quindi con il risultato mostrato in figura 3.7, dove è possibile notare il contorno che circoscrive il municipio ed il suo intorno con la relativa ellisse.

Figura 3.7: Risultati della localizzazione del villaggio

3.3 Schieramento delle truppe

Lo schieramento delle forze offensive viene compiuto sulla base delle analisi effettuate sulla zona di gioco, le quali restituiscono l'area che viene reputata essere la più rilevante ai fini della vittoria.
In questa sezione quindi verrà discusso l'approccio seguito per cercare di massimizzare i risultati in battaglia, tenendo conto delle osservazioni compiute nella fase precedente.
Inoltre è importante ricordare che, come già discusso, il fattore temporale non viene preso in considerazione nella strategia di attacco, motivo per cui tutte le milizie vengono dispiegate il prima possibile in battaglia.

3.3.1 Conformazione delle truppe

Rival Kingdoms permette di utilizzare diverse tipologie di truppe durante uno scontro, lasciando all'utente la decisione della formazione da portare in battaglia.
Ogni truppa ha un certo livello e conta un certo numero di unità: ad esempio durante la raccolta dei dati sono stati utilizzati degli arcieri al livello 3 composti da cinque unità per truppa. Queste informazioni sono rilevanti per l'esito del combattimento, quindi vengono integrate in tutti i record delle battaglie.
Non è stato previsto un automatismo in grado di riconoscere dinamicamente queste informazioni in quanto, se non modificata manualmente, la formazione rimane sempre la stessa. Le truppe impiegate vengono però descritte in un file json così formato:

```
{
  "troops": [
    {
      "name": "dragon",
      "level": 5,
      "count": 1
    },
    {
      "name": "warden",
      "level": 3,
      "count": 5
    },
    {
      "name": "warden",
      "level": 3,
      "count": 5
    },
    {
      "name": "soldier",
```

```
        "level": 3,
        "count": 5
      }
    ]
  }
```

Ogni qual volta si decide di modificare la conformazione è necessario aggiornare anche questo file per continuare a produrre dei record significativi.

3.3.2 Calcolo delle posizioni di schieramento

Le posizioni sulle quali vengono impiegate le forze di attacco vengono calcolate sulla base dell'ellisse prodotta dalla fase di analisi del villaggio, ma mantenendo sempre un certo grado di casualità. Infatti vengono estratti randomicamente degli angoli da 0 a 360°, ognuno dei quali identifica un raggio che parte dal centro dell'ellisse; calcolando l'intersezione tra tale raggio e l'ellisse stessa si ottiene un punto nel quale è possibile schierare una milizia. Il calcolo della lunghezza del raggio dato l'angolo (θ) ed il centro dell'ellisse (a, b) si ottiene dalla seguente formula:

$$r(\theta) = \frac{x_e y_e}{\sqrt{(y_e \cos\theta)^2 + (x_e \sin\theta)^2}}$$

Il valore calcolato viene poi impiegato per trovare le coordinate (x, y) del punto ricercato, tenendo in considerazione anche l'angolazione dell'ellisse α_e:

$$\begin{cases} x = x_e + r(\theta)\cos(\theta + \alpha_e) \\ y = y_e + r(\theta)\sin(\theta + \alpha_e) \end{cases}$$

In entrambe le formule sono state utilizzate angolazioni espresse in radienti, per cui si sono operate conversioni secondo la seguente formula:

$$r^{rad} = \frac{g^\circ \times \pi}{180^\circ}$$

Una volta ottenuto un potenziale punto di schieramento bisogna verificare se effettivamente sia possibile realizzare tale operazione; se il punto risiedesse

all'interno del contorno sarebbe impossibile piazzare una truppa in quel luogo in quanto il gioco impedisce il posizionamento sopra gli edifici. Pertanto, prima di attuare la mossa, viene verificato se il punto calcolato ricada al di fuori del contorno, utilizzando la stessa metodologia impiegata per verificare se lo stronghold sia contenuto nel confine (spiegata nella sezione 3.2.1); questa volta però il posizionamento verrà avviato soltanto se la funzione *pointPolygonTest* restituisce un valore negativo. In caso contrario viene estratto un nuovo angolo e ricalcolato il punto.

Questo procedimento viene ripetuto tante volte quante sono le milizie specificate nel file json descritto nella sezione 3.3.1.
In figura 3.8 si può verificare graficamente quanto descritto: i segmenti in

Figura 3.8: Visualizzazione grafica dei punti di schieramento delle truppe

rosso identificano i raggi calcolati che hanno prodotto una posizione valida mentre quelli di colore diverso sono quelli scartati.

3.3.3 Schieramento in campo

Una volta trovato un punto valido su cui schierare una milizia si può procedere al piazzamento.
Per lanciare una truppa è sufficiente operare una gesture di swipe (descritta in sezione 2.2.3) che parte dall'icona della truppa designata e arriva al punto calcolato precedentemente.
Dal momento che le icone delle truppe vengono posizionate sempre nell'angolo in basso a sinistra della view, i punti di partenza delle gesture sono calcolati staticamente senza l'ausilio della ricerca del template all'interno dello schermo.

3.4 Salvataggio dei risultati

Dal momento in cui tutte le truppe sono state disposte in campo, non resta che attendere la fine dello scontro per poi registrare i risultati ottenuti. Quest'ultimo passo deve però forzatamente essere compiuto attraverso due step differenti in quanto le informazioni che si intendono memorizzare (il risultato della partita e la percentuale di distruzione) sono reperibili tramite due schermate diverse. Conclusa la battaglia è possibile sapere se è stata ottenuta una vittoria o una sconfitta, mentre per la percentuale di distruzione, che non viene riportata nella schermata riassuntiva, è necessario accedere alla sezione dedicata ai risultati delle battaglie.
Alla fine di ogni scontro quindi, tutte le informazioni necessarie sono disponibili e pronte al salvataggio, ad eccezione della percentuale di distruzione. Si è quindi deciso di salvare i record ad ogni partita conclusa, utilizzando dei valori di default per l'informazione sulla distruzione. Una volta conclusa la serie di battaglie (ovvero quando non sono più disponibili pietre di battaglia) si procede all'aggiornamento dei dati.

3.4.1 Salvataggio dell'esito di battaglia

Come anticipato il risultato della partita viene mostrato immediatamente al termine della battaglia. Il riconoscimento dell'esito viene eseguito tramite *cosine similarity*, come descritto nella sezione 2.4.2.
Nelle figure 1.4 e 3.9 troviamo le schermate di vittoria e di sconfitta utilizzate come template per il calcolo del cosine similarity.

Figura 3.9: Esempio di schermata riassuntiva a fine combattimento (sconfitta)

3.4.2 Salvataggio delle percentuali di distruzione

Ad ogni completamento di una serie di battaglie, per poter raccogliere informazioni riguardo le percentuali di distruzione dei villaggi attaccati è necessario tornare alla home, entrare nel menù e successivamente accedere alla sezione *Battle History*, unico punto dell'applicazione dove è possibile visualizzare tali dati.
Lo spostamento tra le varie view viene performato con la combinazione di ricerche interne di immagine (sez. 2.4.1) e tap sullo schermo (sez. 2.2.2).

Una volta arrivati nella sezione di interesse lo schermo si presenterà come in figura 1.5. Di tutta la schermata, l'unica zona di interesse è solamente quella in cui è contenuta la percentuale. In primo luogo, per eliminare gli elementi di disturbo, viene ritagliato un rettangolo contenente soltanto la cifra ricercata; il crop viene estrapolato dalla porzione di schermo immediatamente alla sinistra del simbolo di percentuale (riportato in figura 3.10), del quale vengono ottenute le coordinate tramite la ricerca interna di immagine.
In Python è possibile effettuare un crop semplicemente specificando le coordinate degli angoli del rettangolo da ritagliare:

```
img = img[CROP_Y:CROP_Y+CROP_Y_OFFSET,
    start_x-CROP_X_OFFSET:start_x]
```

Questa linea di codice è quella impiegata nel ritaglio della porzione di interesse, dove le coordinate dell'altezza del rettangolo sono costanti in quanto il dato è sempre posto alla stessa latitudine; possono variare invece le coordinate dell'asse x, a seconda del numero di cifre del quale è composta la percentuale ricercata.

Figura 3.10: Template utilizzato per localizzare la percentuale di distruzione

Viene quindi utilizzata una larghezza fissa, che possa contenere tutte le cifre possibili (nel peggiore dei casi saranno di 3 cifre per raffigurare il valore 100); la fine del frammento di immagine di interesse coinciderà quindi con la coordinata x del punto ottenuto dalla ricerca del simbolo di percentuale, mentre l'inizio sarà uguale alla sottrazione tra il punto di fine e la massima larghezza che può assumere una cifra.
Una volta ottenuta la porzione di immagine ricercata, viene applicato un threshold per marcare più precisamente i contorni delle cifre.

L'ultimo passo viene compiuto utilizzando la libreria pytesseract[1], un wrapper per Tesseract-OCR[17]. *Tesseract-OCR engine* è un riconoscitore ottico di caratteri open source e disponibile su GitHub[2], riconosciuto attualmente come uno dei più accurati motori OCR disponibili.
Con la seguente riga di codice è possibile avviare il processo di riconoscimento:

```
ret_val = pytesseract.image_to_string(img, config='outputbase
    digits')
```

Il parametro principale preso in input dalla funzione è l'immagine da analizzare, mentre in secondo luogo è possibile specificare il tipo di output che si vuole ottenere (in questo caso solo cifre) utilizzando la configurazione addizionale *outputbase digits*.

Tutta l'operazione va ripetuta tante volte quante sono le battaglie portate a termine nell'ultima serie: per scorrere le varie partite è sufficiente inviare un tap sull'elenco di scontri riportato a sinistra, dove vengono ordinati a partire dal più recente.
I tap vengono effettuati staticamente: inizialmente è selezionata l'ultima battaglia in ordine temporale e per passare alla successiva viene inviato un tap sull'elemento seguente sommando ogni volta all'ordinata una costante.

3.4.3 Struttura dei dati

In base agli accorgimenti presi nella fase di attacco per cercare di rendere deterministico l'esito di ogni scontro, è anche doveroso memorizzare tutte le informazioni necessarie per un'eventuale replicazione della battaglia. Queste informazioni potranno poi essere essenziali per allenare reti neurali in grado di migliorare le tecniche di attacco.
In primis, lo screenshot del villaggio è l'elemento essenziale impiegato nella

[1] https://pypi.python.org/pypi/pytesseract
[2] https://github.com/tesseract-ocr/tesseract

parte di apprendimento automatico e punto di partenza di ogni possibile sviluppo futuro.
Anche i dati relativi all'ellisse prodotta in fase di analisi sono molto importanti in quanto necessari per l'effettivo schieramento delle truppe. Per poter replicare una ellisse bisogna conoscere:

- Le coordinate del centro;
- La lunghezza dei semiassi (maggiore e minore);
- L'inclinazione della figura.

Dall'ellisse si possono quindi generare i posizionamenti delle varie forze di attacco schierate: il parametro principale da salvare in memoria è sicuramente l'angolo impiegato per il calcolo della posizione, al quale poi vanno aggiunte le coordinate ottenute e soprattutto le caratteristiche di ogni milizia schierata, quali il tipo, il numero di unità e il livello.
La posizione dello stronghold è importantissima: rappresenta l'obiettivo principale in uno scontro ed è utilizzato anche per la definizione dell'ellisse.
In ultimo, il risultato della battaglia e la percentuale di distruzione ottenuta sono parametri imprescindibili per la fase di training di qualsiasi eventuale rete neurale sviluppabile e rappresentano l'unico modo per poter effettuare una procedura di backpropagation efficace4.2.

Per l'effettivo salvataggio in memoria di queste informazioni è stato deciso di utilizzare il formato Json[1].
Ogni battaglia è identificata univocamente dal path dello screenshot, che avrà un nome univoco identificato da un numero seriale crescente. Di seguito viene riportato un esempio di record:

[1] JavaScript Object Notation

```
...
"camp_screen/camp649.png":{
  "ellipse_center_x":325.7707824707031,
  "ellipse_center_y":169.19024658203125,
  "ellipse_len_x_axis":251.55335998535156,
  "ellipse_len_y_axis":233.29263305664062,
  "ellipse_angle":195.97491455078125,
  "stronghold_level":5,
  "stronghold_x":365,
  "stronghold_y":161,
  "troop1_name":"dragon",
  "troop1_units":1,
  "troop1_level":5,
  "troop1_x":445.28711538344714,
  "troop1_y":138.94689971587158,
  "troop1_angle":150,
  "troop2_name":"warden",
  "troop2_units":5,
  "troop2_level":2,
  "troop2_x":269.50194243465717,
  "troop2_y":271.8308495298511,
  "troop2_angle":283,
  "troop3_name":"warden",
  "troop3_units":5,
  "troop3_level":2,
  "troop3_x":302.86781150006954,
  "troop3_y":53.057779203342406,
  "troop3_angle":63,
  "troop4_name":"soldier",
  "troop4_units":5,
  "troop4_level":2,
  "troop4_x":273.1684333551708,
  "troop4_y":273.6279875279066,
```

```
    "troop4_angle":281,
    "destr_perc":41,
    "battle_result":0
  },
  ...
```

Il record viene appeso al file contenente tutti gli altri dati non appena la battaglia si conclude, includendo anche i risultati ottenuti in battaglia.
Il campo *battle_result* assume valore 1 in caso di vittoria, 0 altrimenti; l'esito viene interpretato secondo quanto detto nella sezione 3.4.1.
L'attributo *destr_perc* invece viene preimpostato con valori di default: 100% in caso di vittoria e 20% per la sconfitta. Una volta esaurite le pietre di battaglia viene avviata la procedura descritta nella sezione 3.4.2 per sostituire i valori predefiniti con quelli reali.
In questo modo avremo sempre dei record completi in memoria che nel giro di pochi minuti verranno aggiornati.

3.5 Iteratività del bot

Il bot, una volta avviato, segue un iter ben definito, cercando di compiere tutte le lotte possibili a seconda della disponibilità di pietre e aspettando la rigenerazione delle stesse prima di ritornare in battaglia.
In particolare, lo script:

1. Riconosce in quale view dell'applicazione si trova, distinguendo tra home, mappa o battaglia già avviata. Prima di iniziare il processo di riconoscimento tenta di chiudere eventuali finestre occasionali quali ad esempio quella del premio giornaliero oppure eventuali promozioni attive per l'acquisto di risorse aggiuntive.

2. A seconda della view nel quale si trova, esegue i passi necessari per avviare una battaglia PvP.

3. Avvia battaglie consecutive fino a quando sono disponibili delle pietre per combattere. Si può avviare la battaglia successiva direttamente dalla shermata riassuntiva della lotta (fig. 1.4), premendo sul bottone *Battle* mostrato in figura 2.4.

4. Una volta esaurite le pietre, ovvero quando il bottone *Battle* non verrà più mostrato, il bot ritorna alla home, passando per la mappa ed evadendo tutti gli eventuali premi conquistati durante le battaglie. Dalla home viene aperto il menù e successivamente lo storico delle battaglie, sul quale poi viene avviata la procedura di aggiornamento delle percentuali di distruzione descritta in sezione 3.4.2.

5. Aggiornate le percentuali ritorna alla mappa e attende la rigenerazione delle pietre per 60 minuti, dopodiché riparte dal punto 1.

Per lo spostamento tra le varie view viene sempre utilizzato il *template matching* (sez. 2.4.1), grazie al quale è possibile trovare i giusti bottoni da premere con un tap.

Capitolo 4

Apprendimento automatico

L'idea originale emersa durante la fase di ideazione del progetto era quella di integrare al bot un meccanismo di apprendimento automatico per le battaglie.
Dopo la prima fase di raccolta dei dati tramite il bot, il secondo step da seguire prevedeva l'implementazione di una rete neurale che amplificasse le capacità del bot stesso.
Il risultato finale preposto sarebbe stato quello di arrivare ad una CNN in grado di restituire in output le posizioni nelle quali poi il bot avrebbe dovuto posizionare le truppe per migliorare i risultati ottenuti in fase di battaglia.
In questo capitolo viene descritto il primo approccio tentato per raggiungere l'obiettivo, nel quale si cerca di allenare una rete neurale in grado di prevedere l'esito di una gara, prendendo in input la mappa e il posizionamento delle truppe su di essa.
Come vedremo, il poco tempo a disposizione non ha permesso di arrivare ad un punto significativo della sperimentazione, considerando anche le difficoltà incontrate in fase di raccolta dei dati.

4.1 Descrizione della rete neurale

La struttura seguita per l'implementazione di questa rete neurale è abbastanza tradizionale e segue la linea della rete LeNet[18] sviluppata da Yann LeCun: vengono utilizzate le tre operazioni principali per questo tipo di rete, ovvero convoluzione, pooling e classificazione. Questo tipo di struttura è infatti rimasta alla base anche delle più recenti architetture sviluppate e si continua a dimostrare adeguata per diversi usi nell'ambito del riconoscimento di immagine.
Come input layer viene utilizzata la concatenazione dell'immagine raffigurante il villaggio attaccato e la dislocazione delle truppe utilizzata in battaglia.

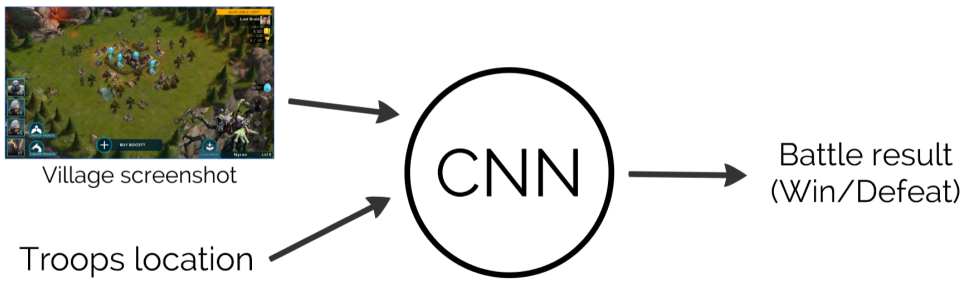


Figura 4.1: Schema della rete neurale di previsione

L'immagine, prima di essere concatenata, viene rimpicciolita fino ad arrivare ad una risoluzione di 128x72 pixel.
Il modello della CNN ideata viene schematizzato in figura 4.4; vengono utilizzati quattro layer convolutivi, dove ogni due viene applicato anche un livello di max pooling.
I layer convolutivi restituiscono tutti un output costituito da 32 filtri, utilizzano un filtro di dimensione 3x3 e una relu come funzione di attivazione. Lo scopo primario di questi livelli è quello di estrarre informazioni rilevanti dall'immagine in input.
I livelli di pooling servono per ridurre le dimensioni dell'output generato dalle

precedenti convoluzioni. In questo caso è stato utilizzato un *max pooling* con filtro di dimensione 2x2, all'interno del quale viene selezionato e mantenuto soltanto il valore maggiore, come rappresentato nell'esempio in figura 4.2.

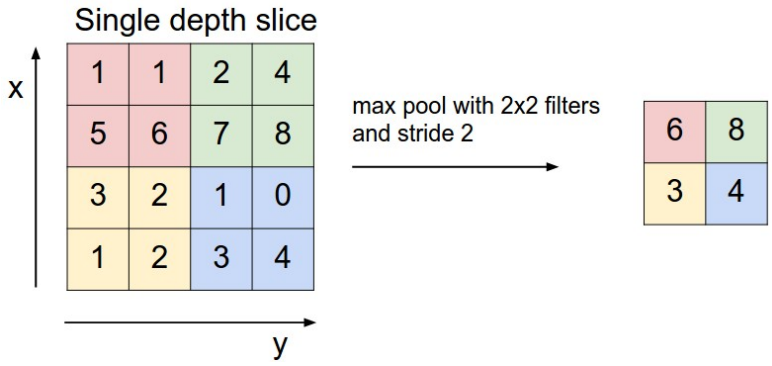


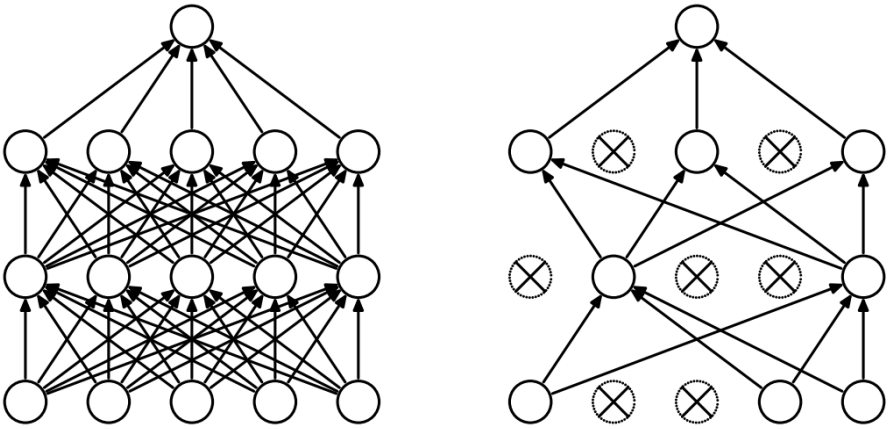
Figura 4.2: Esempio di max pooling con filtro di dimensioni 2x2[1]

Per cercare di prevenire un potenziale overfitting della rete neurale sono stati utilizzati anche due layer di dropout[15] fra un livello convolutivo e il successivo, grazie ai quali vengono spenti randomicamente alcuni neuroni della rete e tutti i relativi collegamenti in entrata e in uscita da essi, per cercare di rendere il modello meno indipendente dai dati (esempio grafico in figura 4.3). Viene poi effettuato un appiattimento dei risultati ottenuti tramite un flatten layer. In ultimo si utilizzano due livelli totalmente connessi (anche detti densi), al fine di impiegare le informazioni estratte in seguito alle convoluzioni per la classificazione (ovvero predire una vittoria o una sconfitta).

4.2 Training

Il processo attraverso il quale una rete neurale apprende è chiamato *backpropagation algorithm*[19]. Questo metodo è uno dei tanti modi attraverso il quale una CNN può essere allenata.
Elemento centrale della backpropagation è il *supervisor*, ovvero una collezio-

[1]Immagine tratta da: `http://cs231n.github.io/convolutional-networks/`



(a) Standard Neural Net (b) After applying dropout.

Figura 4.3: Reti neurali rispettivamente senza e con dropout[15]

ne di dati già classificati attraverso il quale è possibile correggere la CNN. L'approccio utilizzato è proprio quello di apprendere dagli errori commessi. Una rete neurale consiste nell'insieme di diversi nodi, connessi su più livelli; le connessioni tra nodi adiacenti sono pesate. Lo scopo del training è proprio quello di correggere i pesi attribuiti alle varie connessioni.
Inizialmente tutte le connessioni vengono pesate randomicamente. Per ogni input che la rete riceve verrà prodotto un output, che viene poi comparato con l'output atteso, che è già conosciuto. Quando la rete neurale commette un errore si innesca il meccanismo di backpropagation, in base al quale l'errore viene utilizzato per aggiornare i pesi delle connessioni che maggiormente hanno influito nel produrre la previsione. Questo processo viene ripetuto fino a quando la percentuale di errori commessi dalla rete non diventa accettabile.

Lo stesso tipo di processo viene applicato nella rete neurale di previsione descritta precedentemente: il dataset del quale si dispone è già classificato in quanto conosciamo i risultati delle battaglie. Nel training quindi la rete prenderà in input uno screenshot del villaggio e i relativi posizionamenti delle truppe, producendo poi una certa previsione di vittoria o sconfitta attraverso

un valore compreso tra 0 e 1.
Teoricamente quindi, sulla base dell'errore commesso, ad ogni input i pesi della rete neurale vengono aggiornati, perfezionando sempre di più le capacità di previsione della rete stessa.
Sulla base dei risultati ottenuti dal training sarà poi possibile riuscire ad adattare la rete per elaborare in output le posizioni più indicate al conseguimento di una vittoria in battaglia.
La rete quindi non sarà più chiamata a compiere una classificazione, bensì una regressione, tramite la quale si cerca di ottenere in output dalla rete dei valori non classificabili. La regressione è infatti un metodo di stima di valori attesi che variano in funzione di altri valori. Nel caso della rete neurale sviluppata, le disposizioni delle truppe varieranno in funzione dell'input e dell'esito positivo dello scontro ricercato.

4.3 Problematiche riscontrate

Le problematiche che non hanno permesso di arrivare ad un punto stabile in questa parte del lavoro sono correlate alla mancanza di tempo necessario alla sperimentazione. Infatti la fase di raccolta dei record ha richiesto più tempo del dovuto e probabilmente ad oggi i dati raccolti non sono ancora sufficientemente adeguati all'impiego nella rete neurale progettata. Purtroppo l'algoritmo di matchmaking utilizzato dai creatori del gioco non risulta adeguato agli scopi del progetto; infatti, su circa 1000 battaglie effettuate dal bot, quasi il 90% sono state vinte con estrema facilità, con una percentuale di distruzione del 100%.
Osservando lo storico delle battaglie e i relativi avversari si può notare che, nonostante il profilo utilizzato dal bot avesse uno stronghold di livello nove, quello dei nemici invece è spesso soltanto al primo livello. Si possono registrare circa 5/6 avversari di fila con villaggi da affrontare pressoché identici; dopodiché il grado di difficoltà aumenta fino ad arrivare di media ad un livello intermedio (municipio circa al quarto livello) ma anche in questo caso

la battaglia viene ampiamente vinta visto il divario di livello ancora troppo alto. Terminata anche questa battaglia di solito viene selezionato un nemico arduo da sconfiggere, di livello intermedio con difese molto sviluppate oppure di grado molto alto e pressoché imbattibile anche da un utente reale.
Questo metodo di accoppiamento purtroppo non consente di avere una raccolta di dati omogenea e, soprattutto, che permetta alla rete neurale di apprendere dagli errori. Infatti, in base ad una prima rete neurale sviluppata per prevedere l'esito della battaglia, si può notare come ci sia una repentina saturazione in fase di training, con la funzione di loss che si stabilizza quasi immediatamente su valori alti. Difatti dopo il training la rete restituisce sempre esito positivo, per qualsiasi input ricevuto.
Per cercare di compensare lo squilibrio dei dati è stato tentato un esperimento: assegnare dei pesi diversi ai vari record, prediligendo le sconfitte. Questo procedimento, applicato in fase di fit del modello, ha lo scopo di suggerire alla CNN di porre molta più attenzione sulle sconfitte piuttosto che sulle vittorie. Nello snippet di codice seguente viene definita la class weight utilizzata:

```
cw = { # class_weight
    0: wins,
    1: defeats
}
```

Sostanzialmente è stato dato un peso pari al numero di vittorie per i record in cui si era ottenuta una sconfitta, mentre ai record di vittoria è stato assegnato un peso molto inferiore, pari al numero complessivo di partite perse.
Questa prova però non ha evidenziato sostanziali cambiamenti favorevoli.
Inoltre per cercare di irrobustire la rete, è stato aggiunto un certo grado di randomicità alle immagini, utilizzando un ulteriore preprocessing:

```
datagen = image.ImageDataGenerator(
featurewise_center=False,
featurewise_std_normalization=False,
rotation_range=10,
```

```
    width_shift_range=0.05,
    height_shift_range=0.05,
    horizontal_flip=False)
```

Gli screenshot dei villaggi vengono leggermente ruotati e traslati randomicamente prima di essere dati in input alla rete; non si sono però riscontrati giovamenti neanche da tale operazione.
Un altro possibile tentativo potrebbe essere quello di ignorare la gran parte delle battaglie con vittorie schiaccianti: esperimento purtroppo non concretizzabile attualmente in quanto i tempi di raccolta dei dati lieviterebbero consistentemente a causa delle tante pietre di battaglia impiegate per combattimenti che poi non andrebbero presi in considerazione.
Si è inoltre tentato di sviluppare al massimo il villaggio, in particolar modo lo stronghold, escludendo del tutto i potenziamenti delle forze di attacco, per cercare di incentivare l'algoritmo di matchmaking a selezionare avversari più forti, ma purtroppo non si è arrivati a nessun miglioramento reale.
Di fatto quindi attualmente la rete neurale persiste nel prevedere sempre e comunque una vittoria.

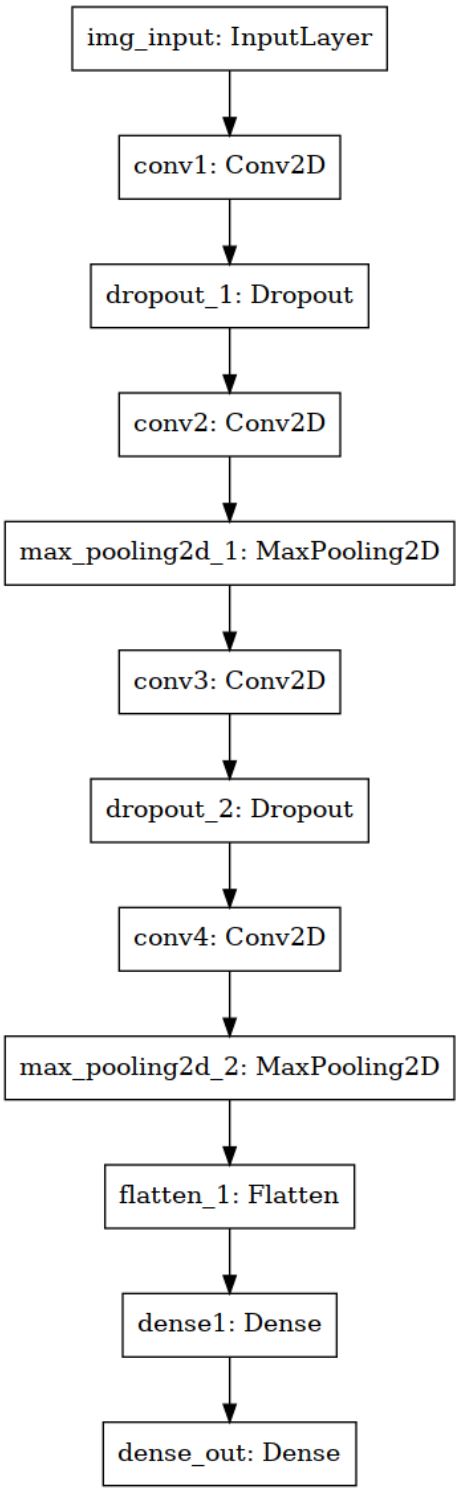


Figura 4.4: Modello della rete neurale di previsione

Conclusioni

In questo lavoro viene introdotto un bot per videogiochi che si differenzia da altri progetti affini per la sua capacità di interagire con un device esterno, emulando l'utilizzo che ne farebbe un normale utente.
Si sono raggiunti dei risultati soddisfacenti per l'implementazione di tale script, che attualmente, se connesso con un telefono o un emulatore, può essere avviato come demone e portare a termine battaglie registrando i risultati ottenuti.
Rappresentano invece un problema i numerosi elementi di disturbo contenuti all'interno dei villaggi, come ad esempio gli alberi, che vengono rilevati e considerati come normali costruzioni quando invece non influenzano in nessun modo le sorti della partita.
Tuttavia il codice si dimostra abbastanza solido ed in grado di gestire la gran parte delle situazioni incontrollate quali per esempio pubblicità interne dell'applicazione o apertura di forzieri contenenti premi extra ottenuti durante le battaglie.
Per quanto riguarda invece l'integrazione di meccanismi di apprendimento automatico, non è stato possibile raggiungere dei risultati appaganti.
Si sono infatti riscontrati problemi dovuti sia al troppo poco tempo impiegato per la raccolta dei dati, sia per l'algoritmo di matchmaking impiegato dai creatori del gioco utilizzato, che purtroppo non favorisce lo scopo del progetto.
In ogni caso i problemi emersi non rappresentano un vero e proprio blocco in termini di sviluppo e potranno essere sicuramente oltrepassati, lasciando

spazio a diversi possibili futuri sviluppi.

Sviluppi futuri

I risultati raggiunti finora lasciano ampio margine di miglioramento ed aprono a possibili futuri scenari, soprattutto per quanto riguarda l'integrazione di reti neurali nell'elaborazione della strategia di battaglia.
Sarà possibile impiegare più tempo per cercare di migliorare il pool di dati al quale accederanno le reti neurali, raccogliendo più informazioni e soprattutto offrendo la possibilità di selezionare le battaglie più rilevanti ai fini dell'apprendimento, scartando invece le meno interessanti.
Una volta espanso il dataset sarà così possibile ottenere risultati più soddisfacenti per quanto riguarda la previsione delle battaglie.
In secondo luogo sarà possibile sviluppare una seconda rete neurale che, sulla base dei pesi elaborati dal training della precedente, riesca a produrre le posizioni più indicate nelle quali schierare le truppe, al fine di sconfiggere anche avversari di livello più alto.
Inoltre, per quanto il bot sembri funzionare adeguatamente, è ancora migliorabile sotto diversi punti di vista.
Infatti, per quanto riguarda l'analisi del villaggio avversario sarebbe opportuno ottenere un'ellisse calcolata sulla base non solo del contorno contenente lo stronghold, bensì anche della posizione del municipio stesso.
Sarà sicuramente necessario anche cercare una soluzione per ignorare gli elementi di disturbo contenuti nei villaggi avversari, come ad esempio rocce o alberi.
Si potrebbe cercare di aumentare la compatibilità del bot con i diversi device, estendendo le risoluzioni supportate e gli aspect ratio delle nuove generazio-

ni di dispositivi che montano display in 18:9. In ultimo, il modo con cui viene riprodotto il pinch out (ampiamente discusso nella sezione 2.2.4), seppur efficace risulta molto macchinoso e fortemente dipendente dal dispositivo sul quale viene riprodotto. Si potrebbe cercare di produrre una sequenza di eventi più generali e applicabili su tutti i dispositivi, oppure implementare un metodo alternativo che però, fino ad ora, non è ancora stato trovato.

Bibliografia

[1] K. J. Millman and M. Aivazis. Python for scientists and engineers. *Computing in Science Engineering*, 13(2):9–12, March 2011.

[2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[3] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001. [Online].

[4] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30, March 2011.

[5] François Chollet et al. Keras. `https://github.com/keras-team/keras`, 2015.

[6] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer vision and pattern recognition (CVPR), 2012 IEEE conference on*, pages 3642–3649. IEEE, 2012.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[8] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE*

conference on computer vision and pattern recognition, pages 3431–3440, 2015.

[9] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279, 2013.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[11] Michelle McPartland and Marcus Gallagher. Creating a multi-purpose first person shooter bot with reinforcement learning. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 143–150. IEEE, 2008.

[12] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, pages 1–8. IEEE, 2016.

[13] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *AAAI*, pages 2140–2146, 2017.

[14] Andrea Asperti, Carlo De Pieri, and Gianmaria Pedrini. Rogueinabox: an environment for roguelike learning. *International Journal of Computers*, 2, 2017.

[15] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[16] Hieu V Nguyen and Li Bai. Cosine similarity metric learning for face verification. In *Asian conference on computer vision*, pages 709–720. Springer, 2010.

[17] Ray Smith. An overview of the tesseract ocr engine. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, volume 2, pages 629–633. IEEE, 2007.

[18] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. In *The handbook of brain theory and neural networks*, pages 255–258. MIT Press, 1998.

[19] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591. IEEE, 1993.