UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA

FACULTATEA DE MATEMATICĂ SI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

LUCRARE DE LICENȚĂ

Generarea procedurala a mediului înconjurător în jocuri folosind rețele neuronale recurente

Conducător științific

Mircea Ioan-Gabriel,

Asistent

Absolvent

Ivanov Silviu-Gabriel

2018

Abstract

De cel puțin 30 de ani s-a folosit în cadrul jocurilor generarea procedurală de conținut. De curând un nou domeniu se face cunoscut, cel de machine learning, prin abordările pe care le aduce în generare conținutului fără a fi nevoie de specificarea unor reguli stricte. Acest document prezinta o analiză a generării procedurale a mediului înconjurător în jocuri folosind paradigme relativ noi în contrast cu metodele bazate pe construcție, căutare sau rezolvare. Procesul va consta în crearea unei rețele neuronale recurente de tipul LSTM[1] ce va fi antrenată pe conținut deja existent din cadrul jocului, ca mai apoi modelul creat să poată genera nivele noi.

[1] Long Short-Term Memory

Cuprins

Introducere

1. Generare procedurală de conținut

Generarea procedurala de conținut reprezintă o metodă prin care putem sa generam date folosind un algoritm. În decursul anilor importanta a PCG[2] pentru dezvoltarea jocurilor, dar și pentru toate tipurile de conținut a crescut considerabil, crescând totodată și numărul de cercetări în legătură cu acest domeniu, încercând să se descopere moduri noi de a genera conținut de înaltă calitate și dând la o parte interacțiunea umană.

Această metodă este foarte răspândită în toate domeniile de activitate, putând fi generate imagini, muzică precum și obiecte 3D, o importanță puternică având de asemenea și în cadrul sintetizării vocale. Reprezentând o metodă artificiale ce este capabilă să reproducă discursul uman, sintetizarea vocală joacă un rol important în multe sisteme precum: Apple[3], AmigaOS, Microsoft Windows, Atari etc.

Câteva avantaje ce sunt prezente în momentul în care generăm conținut procedural sunt: minimizarea spațiului necesar de stocare a datelor, posibilitatea de a crea un volum considerabil de conținut și abilitatea de a avea o nouă perspectivă asupra întregului produs final.

[2] Procedural Content Generation

[3] Apple a avut primul sistem de operare ce conținea sintetizare vocală.

1.1. Generarea procedurală de conținut în jocurile video

În decursul anilor PCG în jocuri s-a dezvolt foarte mult, utilizând domeniul dezvoltării de jocuri și cercetările tehnice din acest domeniu. Valoarea pe care o aduce este reprezentată de către reducerea costului și a efortului de producție, economisirea spațiul necesar pentru stocarea datelor precum și crearea unui design inovator. Câțiva cercetători academici din domeniul PCG au luat în considerare aceasta provocare și de asemenea au analizat cum generarea de content într-un mod procedural poate sa confere noi experiențe jucătorului și să se adapteze pe placul acestuia. Construind un model formal ei au reușit să modeleze creativitatea computațională și să sporească înțelegerea noastră față de designul jocului [1].

Multe dintre aplicabilitățile „constructive" ale PCG în industria de jocuri, sunt reprezentate de către algoritmi bazați pe zgomot sau gramatici, cu scopul de a crea într-un mod continuu conținut fără a fi necesară o evaluare ulterioară, în timp ce alții se axează pe metode bazate pe rezolvare [2] sau căutare [3]. Toate aceste metode care generează conținut au în comun parametri, constrângeri, algoritmi și obiective create de către proiectanți și cercetători. Chiar dacă noi ne putem inspira din jocul actual, aceste metode bazate pe AI[4] sunt foarte rar folosite pentru a genera conținut singure. Conținutul ce va fi generat poate aparține oricărui tip din cadrul jocului, de la obiecte din inventar, modelele caracterelor și mediul înconjurător până la misiuni, reguli și arme.

Trebuie menționat ca în cadrul acestei lucrări ne vom focusa atenția asupra părții funcționale[5] ale jocului, și nu pe design sau partea artistică a acestuia. Un exemplu de conținut ce va fi exclus îl reprezintă atributele cosmetice ale diferitelor obiecte deoarece ele nu afectează într-un mod direct acțiunile jucătorului. De asemenea trebuiesc precizate diferențele cheie între generarea procedurală de conținut în general, în toate domeniile, și PCG în jocuri. În timp ce în alte arii de cercetare ale PCG avem posibilitatea de a crea orice tip de conținut fără a fi constrânși în vreun fel, în ceea ce privește aria jocurilor suntem strict constrânși de limitări precum modul în care funcționează și regulile jocului – *ergodic*[6] *media* [4]. Un nivel ce conține o structură sau un număr de inamici ce

[4] Artificial Intelligence

[5] Prin conținut funcțional ne referim la modificări care vor avea un efect sporit asupra experienței jucătorului.

[6] Literatura ergodică necesită un efort netrivial pentru a da posibilitatea cititorului să traverseze textul.

conduc jucătorul către o fundătură, sau fac imposibilă finalizarea nivelului, nu sunt acceptabile, chiar dacă aranjamentul noului conținut este nou și atractiv, va conduce la experiența negativă a jucătorului și va fi chiar mai rău decât un nivel care este terminabil dar este creat de la început si stocat în memorie. De exemplu *Figura 1* reprezintă un labirint în care se poate doar intra, nu și ieși, un astfel de rezultat cu siguranță ar conduce la o experiență neplăcută a jucătorului. Desigur ca vor exista mereu provocări și în cadrul celorlalte domenii ce nu au legătură cu jocurile, provocări ce vor avea atașate asupra lor tot felul de constrângeri cum ar fi: a crea o imagine ce pare a fi reală; totuși în această lucrare ne vom focusa doar pe ce este din domeniul jocurilor.

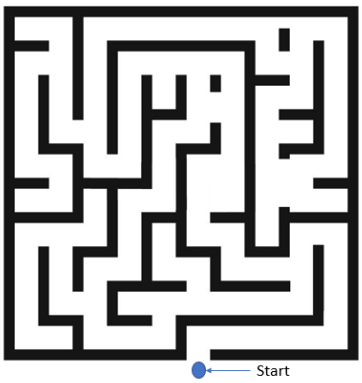


Figura 1 - Labirint fără ieșire

Unul dintre pionierii acestui domeniu este *Dwarf Fortress,* un joc în cadrul căruia este simulată construcția și managementul unei fortărețe. Grafica jocului este bazată pe text, jocul neavând un sfârșit sau un obiectiv principal ce trebuie îndeplinit. Un atu important al acestui joc îl constituie modul în care lumea este generata. Procesul implică generarea procedurală a elementelor de bază precum circuitul de drenare, temperatura, distribuția mineralelor, elevația și ploaia. În *Figura 2* se observă modul în care harta este generată, fiecare joc începând cu acest proces. După câteva minute lumea este populată și istoria începe sa se creeze în funcție de parametrii aleși [5].

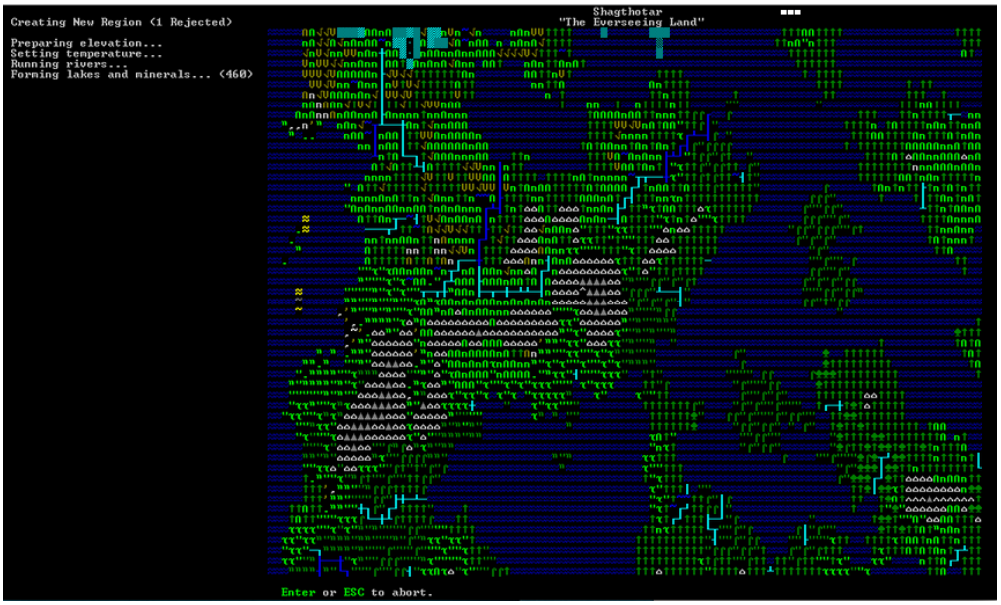


Figura 2 - Exemplu de generare a lumii in Dwarf Fortress [6]

1.2. Cazuri de utilizare pentru PCG

În cadrul acestui capitol vom analiza câteva întrebuințări ale generării procedurale de conținut. Ne vom axa pe avantajele funcționale dar și pe cele economice ale acestei metode.

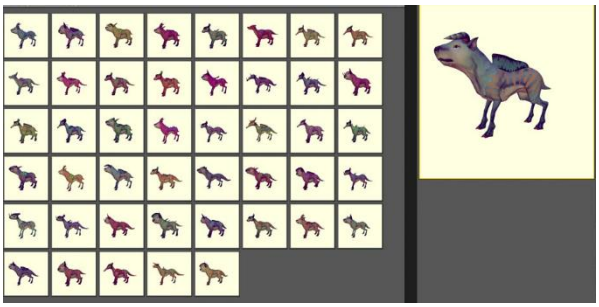


Figura 3- Modele de animale din No Man's Sky

A) Comprimarea datelor

Una dintre primele motivații ale implementării PCG în primele jocuri, dar valabilă și astăzi în jocurile moderne a fost necesitatea comprimării spațiului necesar de stocare a datelor. Neavând suficient spațiu pe disk pentru a putea salva toate nivelele necesare sau toate hărțile unui joc, adăugând faptul că unele jocuri au loc în universuri vaste, producătorii au avut nevoie de o modalitate de a putea genera surplusul de date pentru a fi mai departe trimise către jucător. Jocuri precum *No Man's Sky* [7] ce se desfășoară într-un univers aproape nelimitat, ar fi imposibile de salvat pe disk, astfel dezvoltatorii au folosit PCG pentru a crea plante și animale, salvând pe disk doar date absolut necesare precum proprietăți și caracteristici ale mediului înconjurător. În *Figura 3* putem observa cum dintr-un model de bază sunt generate modele altor animale din cadrul jocului.

B) Cooperare în creativitate și design

Alt factor important în procesul de dezvoltare a jocurilor este reprezentat de către design. Generarea procedurală folosind *Machine Learning* poate ajuta un designer uman să creeze un conținut mult mai inedit. Partea „co-creativă[7]" vine din faptul că cel ce proiectează poate antrena un model punându-i la dispoziție exemple din domeniul respectiv. Procedura poate reduce drastic timpul dedicat pentru antrenarea creatorului, de asemenea poate micșora gradul necesar de pregătire pentru acesta deoarece va avea mult mai puțină treabă de făcut. Totodată bugetul proiectului va fi redus, deoarece numărul necesar de angajați va fi redus considerabil. În cele din urmă, PCGML[8] poate fi folosit și pentru auto-completarea nivelelor și a conținutului ce a fost specificat doar parțial de către dezvoltator, dar nu încă finalizat.

[7] Co-creație reprezintă o strategie prin care diferite entități participă în procesul de creație cu scopul de a o face împreună mai bine decât ar face-o fiecare pe cont propriu.

[8] Procedural Content Generation using Machine Learning

C) Reparare

O altă funcționalitate pe care PCGML o oferă este posibilitatea de a identifica și repara zonele care nu sunt jucabile sau a oferi sugestii cum ar putea fi reparate. O metodă folosită adeseori o reprezintă cea a autoencoderilor[9] capabil să refacă și să repare porțiuni din conținut precum și segmente corupte.

D) Analiză și evaluare

Principala diferență între PCGML și PCG o reprezintă capabilitatea de a recunoaște, evalua și analiza conținutul jocului. Creând un model bazat pe nivelele originale ale un specific joc, sau pe conținutul original, modelul va putea mai târziu să identifice conținutul ce a fost creat de către un algoritm față de cel creat de către un proiectant sau un jucător. În *Figura 4* ce aparține unei cercetări cu privire la repararea imaginilor [8] se observă cum diferite abordări reușesc să rafineze și să clarifice o porțiune din imagine.

E) Generare autonomă

Ultimul dar nu cel din urmă caz de utilizare a PCG este posibilitatea de a crea în mod continuu conținut fără a fi nevoie de interacțiune umană. Această funcționalitate este adeseori întâlnită în mai toate câmpurile de cercetare în ceea ce privește inteligența artificială. Generarea autonomă conferă posibilitatea de crea și oferi conținut *online*[10]. Este de dorit pentru un joc să poată genera conținut în timp ce rulează, fără a fi nevoie o ajustare de către un om, sau să încarce nivelul/harta/conținutul de pe disk.

Figura 4- Clarificarea unui segment al imaginii [8]

[9] Rețea neuronală artificială folosită pentru a învăța un model o modalitate eficientă de comprimare a datelor.

[10] Acumularea informațiilor și prelucrarea lor în același timp care altcineva interacționează cu ele și le modifică/

2. Inteligența artificială

Inteligența artificială sau AI reprezintă un domeniu de studiu în cadrul căruia se încearcă înțelegerea entităților inteligente. AI momentan cuprinde o varietate de subdomenii, de la cerințe specifice precum jucatul de șah, scriere de poezii, diagnosticarea bolilor sau demonstrarea teoremelor matematice până la zone mult mai generale precum percepția și raționamentul logic. Adeseori oamenii de știință din cadrul acestui domeniu vast tind să-și aplice metodele și algoritmii în orice altă zonă care necesită efort intelectual uman, demonstrând astfel universalitatea aplicabilității acestui domeniu [9].

Ce înseamnă a fi inteligent și cum putem noi testa acest lucru? Propus în anul 1950 de către Alan Turing[11], test ce îi poartă și numele, *„The Turing Test”* a fost conceput pentru a aduce o definiție satisfăcătoare a inteligenței computaționale. Testul constă în capacitatea calculatorului de atinge performanțe cognitive umane cu scopul de a fi capabil să inducă în eroare un interogator uman. Câteva capabilități necesare ale computerului pentru a trece cu succes acest test ar fi: *procesarea naturală a limbajului*, să comunice și să stăpânească cu desăvârșire o anumită limbă; *reprezentarea informațiilor,* capabilitatea de a stoca informații în timpul unei interogări; *machine learning*, să se adapteze noilor circumstanțe și în cele din urmă *raționament automat*, să fie capabil să folosească informațiile stocate să trag noi concluzii sau să răspundă la întrebări.

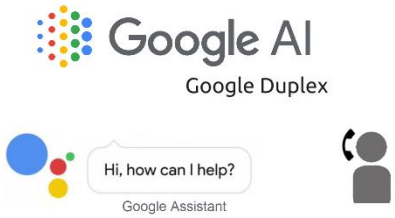


Figura 5 - Google Duplex

Figura 5 face parte din cadrul prezentării Google din data de 8 Mai 2018, unde s-a prezentat o noua tehnologie numită *„Google Duplex”* capabilă să întrețină conversații sofisticate într-o manieră complet autonomă, cu scopul de a crea o rezervare. Deși se recunoaște că sistemul nu poate finaliza autonom rezervări ce au un grad ridicat de dificultate, din punctul meu de vedere această tehnologie se află printre primele care sunt capabile să treacă testul turing într-o conversație naturală.

[11] Personalitate de mare influență în domeniul informaticii, fost matematician, logician si filosof, reprezintă unul dintre părinții inteligenței artificiale.

2.1. Machine Learning

Machine Learning reprezintă adeseori modificările aduse unui sistem ce se ocupa cu îndeplinirea unor cerințe din domeniul inteligenței artificiale. Deși sistem AI se ocupă cu observarea și modelarea mediului înconjurător cu scopul de a determina acțiunile corecte ce trebuiesc făcute într-un mod autonom fără a fi nevoie de interacțiune din exterior, nu înseamnă neapărat ca acesta este inteligent [10]. Vom analiza în ceea ce urmează cum ajustarea câtorva componente ale acestui „*agent"* sunt necesare în anumite situații pentru ca acesta să poată fi numit inteligent.

- Posibilitatea de a se adapta la tot ce apare nou. Cum lumea este într-o continuă mișcare și evoluție nu ar fi practic să trebuiască sa adaptam și să remodelăm de fiecare dată sistemul, astfel, implementarea machine learning-ului devine foarte utilă pentru a face față fluxului continuu de modificări.
- Am dori ca *„agentul"* să fie capabil să-și ajusteze singur structura internă pentru a ajunge la rezultatul dorit, în funcție de exemplele pe care i le furnizăm. Astfel, într-un număr relativ mare de iterații și de exemple procesate acesta este capabil să se aproprie foarte mult de rezultat, chiar dacă datele de intrare sunt noi pentru acesta.
- De multe ori se întâmplă ca mașinăriile dezvoltate să nu aibă o structură potrivită pentru mediul în care activează fie din cauza unor greșeli logice, fie funcționale. În această privință se pot folosit diverse metode din cadrul machine learning-ului pentru a ajut la designul acestor mașinării.

Astăzi interesul pentru domeniul machine learning-ului este într-o continuă creștere, în mare parte el se datorează nevoii de crea modele capabile să folosească seturi de date deja existente cu pentru a se antrena. Una dintre cele mai comune abordări o reprezintă folosirea de *DNN*[12] în cadrul *Deep Learning-ului* [11], acesta fiind potrivit pentru o varietate de sarcini unele dintre ele fiind chiar și generarea de conținut precum: imagini, videoclipuri și înregistrări audio.

[12] Deep Neural Network

2.2. Rețele neuronale artificiale

Considerăm o rețea neuronală artificială ca fiind un model simplificat al structurii rețelei neuronale biologice. Un *ANN*[13] este format din unități de procesare interconectate. Modelul general al acestor unități de procesare este format dintr-o componentă de însumare și una de returnare a rezultatului. [12] Componenta ce are ca scop însumarea primește *N* valori ca și date de intrare, după care atribuie fiecăruia dintre ele câte o greutate și în cele din urmă calculează suma lor. A doua componentă ia această *valoare de activare* și o returnează sub forma unui *semnal*. În funcție de semnul pe care îl are fiecare greutate se poate spune dacă acel input este *excitator* sau *inhibitor* [13]. De asemenea, inputurile și outputurile pot fi discrete sau continue. În *Figura 6* se poate observa modul de funcționare a unui neuron. Acești neuroni sunt grupați pe straturi, rețeaua fiind formată dintr-unul sau mai multe straturi conectate între ele. În cele mai multe cazuri rețea conține un strat de intrare, unul de ieșire și câteva straturi între cele două numite straturi *ascunse*.

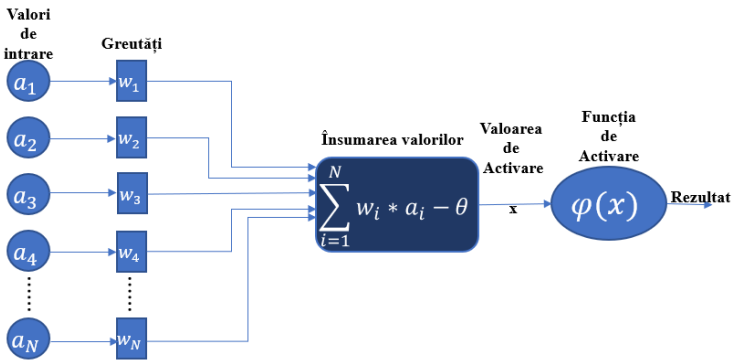


Figura 6 - Modelul perceptronului unui neuron

Una dintre cele mai populare metode de învățare este reprezentată de *backpropagation.* Acest proces implică propagarea erorilor înapoi plecând de la stratul de ieșire către straturile ascunse cu scopul de a recalcula greutățile pentru unitățile de procesare din cadrul acestor straturi. Eroarea este calculată folosind diferența între rezultatul dorit și rezultatul obținut pentru fiecare unitate de ieșire.

Cea mai simplă rețea neuronală este cea în care neuronii de pe un strat comunică doar cu cei de pe stratul următor (*feedforward*), informația călătorind de la intrare spre ieșire. Tot foarte populară este și rețeaua neuronală recursivă, unde datele merg în mai multe direcții. Această rețea posedă o abilitate ridicată de învățare fiind adeseori folosită scopuri mult mai complexe cum ar fi învățarea scrisului de mână sau chiar recunoaștere vocală.

[13] Artificial Neural Network

2.3. Rețele neuronale recursive

Rețeaua neuronală recursivă este un tip de rețea neuronală artificială care operează cu structuri secvențiale de date. De obicei, într-un ANN datele de input și cele de output sunt independente una față de cealaltă, dar în cazul multor situații acesta nu este un comportament dorit. De exemplu dacă dorim să deducem următorul cuvânt dintr-o secvență de cuvinte este necesar să ținem cont de cuvintele care au fost înainte. RNN sunt *recurente* deoarece ele execută aceeași operație pentru toate elementele dintr-o secvență, având output-ul dependent față de calculările anterioare. O altă metodă prin care putem descrie RNN este să spunem că acestea dețin o „memorie" ce înregistrează informațiile calculate până la o anumită unitate de timp [14].

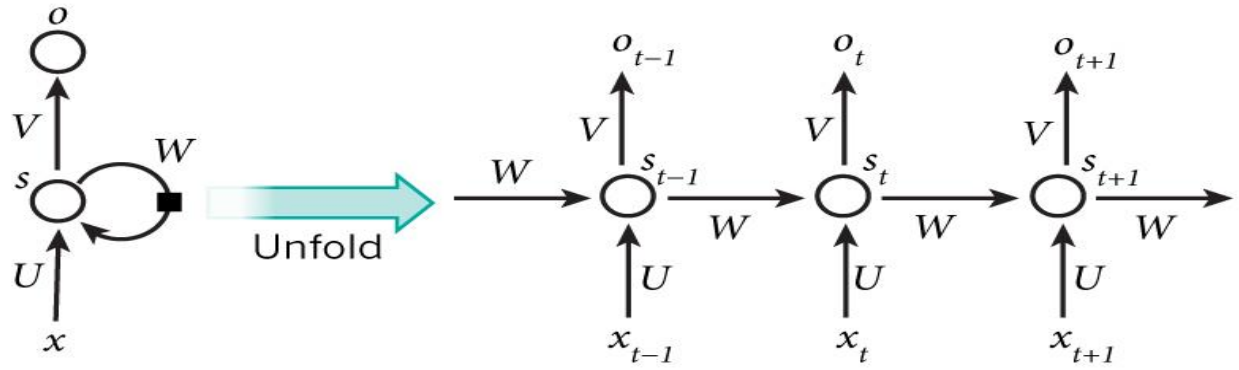


Figura 7- Rețea neuronală recursivă desfășurată [15]

În *Figura 7* se poate observa o rețea neuronală recursivă simplă desfășurată în funcție de momentul la care s-a procesat fiecare input. Dacă am fi studiat de exemplu predicția unui cuvânt și am fi avut cate o propoziție de 10 cuvinte, rețeaua ar fi putut fi reprezentată desfășurat in 10 straturi, câte un strat pentru fiecare cuvânt. În cele ce urmează vom explica notațiile de mai sus precum și câteva formule specifice acestei structuri:

- s_t reprezintă starea interioară a rețelei la un anumit pas *t; reprezintă „memoria"* rețelei, el este calculat în funcție de starea anterioară și input-ul curent $s_t = f(Ux_t + Ws_{t-1})$. Funcția f este adeseori o funcție non-liniară precum *ReLU* sau *tanh.*
- x_t reprezintă data de intrare la un anumit moment *t.*
- o_t reprezintă output-ul la un anumit moment *t.* De exemplu, dacă am dori să prezicem următorul cuvânt într-o propoziție acest output are reprezenta un vector de probabilități pentru fiecare literă din alfabet. $o_t = softmax(Vs_t)$

2.4. LSTM

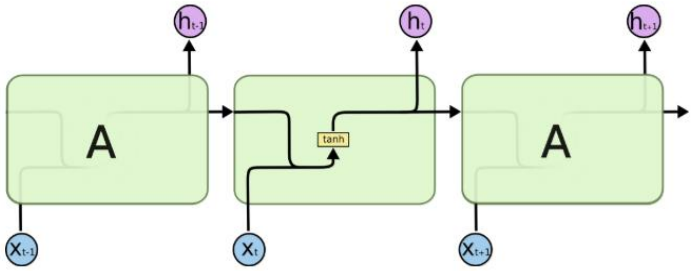


Figura 8 - RNN desfășurat

Ne punem întrebarea dacă, este îndeajuns să avem acces la un număr finit de momente anterioare pentru a putea determina cu succes pe următorul. Răspunsul este că depinde. Câteodată ne este îndeajuns să ne uităm la datele recente pentru a putea procesa cu succes task-ul curent. De exemplu: dacă dorim sa prezicem ultimul cuvânt în „culoarea sângelui este *roșie*", nu avem nevoie de mai multe informații decât cele pe care le deținem deja din contextul curent – este destul de evident, cuvântul căutat este *„roșie"*. În aceste cazuri în care informația căutată se află la mică distanță de informația relevantă nu avem nevoie de o structură mai complexă decât un RNN. Dar, sunt și cazuri în care cantitatea de context necesară este mult mai mare. Să considerăm că rețeaua noastră trebuie să prezică ultimul cuvânt din următorul text „La vârsta de 8 ani m-am mutat împreună cu ai mei în China ... am reușit să ne înțelegem deoarece vorbeam fluid *chineza*". Informația anterioară sugerează că urmează numele unei limbi, dar pentru a ne da seama avem nevoie de informații ce se află cu mult în urmă. Din păcate, cu cât diferența între informațiile relevante și informația curentă se mărește, posibilitatea ca RNN să reușească să conecteze informațiile intre ele cu succes, scade drastic [16].

2.4.1. Descriere

Rețelele LSTM sunt o clasă a RNN capabile să învețe dependințe de lungă durată. Toate rețelele neuronale recurente au forma unor module ce sunt legate între ca și un lanț (*Figura 8*) . Deși rețelele LSTM moștenesc această structură inter modulară, ele vin cu ceva nou în ceea ce privește structura

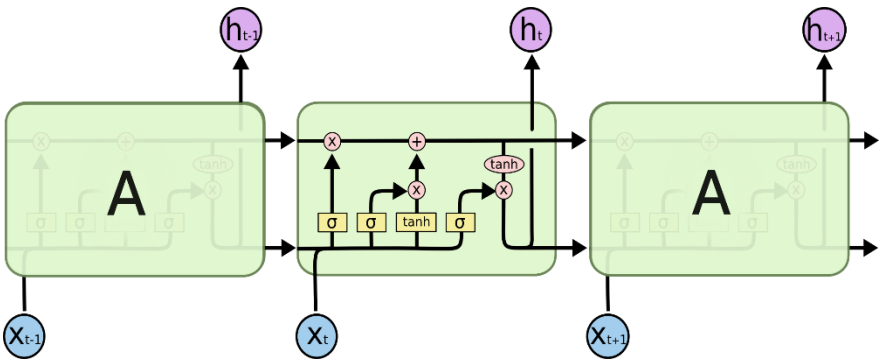


Figura 9 - Rețea LSTM desfășurată

internă a fiecărui modul. În loc de o rețea neuronală cu un singur strat, LSTM conține patru straturi ce comunică intre ele fiecare având un scop bine definit (*Figura 9*).

2.4.2. Modul de funcționare

În *Figura 9* fiecare linie reprezintă un vector de valori, călătorind de la input-ul unui nod până la output-ul acestuia și input-ul următorului nod. Cercurile cu roz reprezintă operații pe vectori și dreptunghiurile galbene simbolizează straturi neuronale. Starea internă a celulei este reprezentată de linia din partea superioară, la ea se adaugă sau se șterg informații la trecerea prin porți.

1. Uitarea – primul pas îl reprezintă filtrarea informațiilor din starea celulei. Decizia se face pe baza unui strat cu o funcție de activare sigmoidală ce calculează un număr între 0 și 1 pentru fiecare element din vectorul de stare al celulei. (0 – uită elementul, 1 – reține elementul). $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
2. Memorarea – al doilea pas constă în adăugarea noilor informații în starea celulei. Acest proces este format din două părți. Primul strat cu activare sigmoidală decide ce valori vom actualiza în starea curentă, în timp ce al doilea strat cu activare *tanh*[14] pregătește noi candidați ce ar putea fi adăugați. Rezultatele formează vectorul ce va actualiza starea internă. $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_f); \tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$
3. Actualizarea stării celulei – al treilea pas constă în actualizarea celulei folosind valorile de la pașii anteriori. Înmulțim rezultatul de la pasul 1 cu starea curentă a celulei pentru a decide ce vom uita, după care adăugam produsul valorilor de la pasul 2 pentru a memora noua informație și a o actualiza pe cea veche. $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$
4. Returnarea rezultatului – ultimul pas îl constituie calcularea și returnarea rezultatului. Primul strat cu activare sigmoidală decide ce elemente din starea internă a celulei vom returna, în timp ce al doilea strat cu activare *tanh* pregătește elementele din starea internă a celulei, aducându-le între valorile 0 și 1. La final cele două rezultate se înmulțesc între ele. $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$; $h_t = o_t * \tanh(C_t)$

[14] Funcție tangentă hiperbolică cu proprietatea că rezultatul ei se află in intervalul (-1,1). Este adeseori folosită ca și funcție de activare.

3. Benchmarks

Deoarece analiza și compararea rezultatelor în ceea ce privește generarea de conținut în jocuri se rezumă de multe ori la subiectivism, vom analiza cele mai bune tehnici folosite în PGML din literatură, grupându-le în funcție de modul de reprezentare a datelor precum și metoda de antrenare aleasă.

Metoda de antrenare

Se vor lua în considerare câteva tehnici din machine learning potrive pentru crearea și antrenarea unui model ce va fi ulterior folosit pentru generarea conținutului în jocuri, acestea sunt : *factorizarea matricelor, maximizarea așteptărilor, numărarea frecvențelor* și *propagarea înapoi.*

Reprezentarea datelor

Fiecare nivel/hartă/element va fi reprezentat într-o manieră ce facilitează metoda de antrenare aleasă, acestea fiind: *grilă, secvențial* și *graf.* Este posibil ca pentru același tip de joc să avem mai multe reprezentări. Analizând fiecare mod posibil de a reprezenta datele, putem observa și compara principalele diferențe, astfel înțelegând îmbunătățirile pe care le aduce fiecare metodologie.

3.1. Reprezentare Secvențială

Fiind una dintre cele mai populare metode de a reprezenta conținutul unui joc de tip platformă. Un joc potrivit pentru această secțiune este „*Super Mario Bros”* [17] *(Figura 10*). De asemenea, oricare alt joc al cărui conținut ajunge la utilizator într-un mod secvențial este o alegere bună.

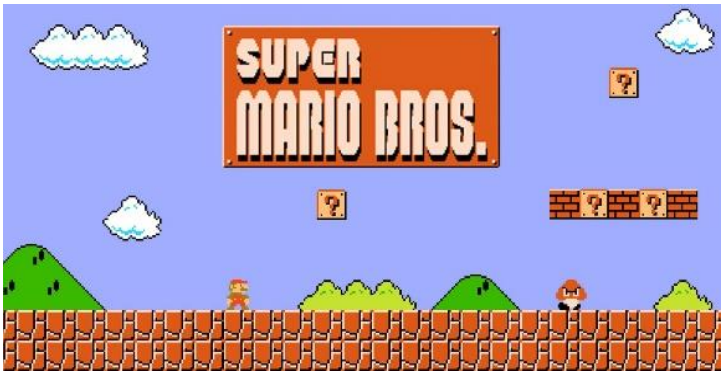


Figura 10 - Super Mario Bros publicat în 13 septembrie 1985 de către Nintendo Creative Department

i) Numărarea frecvențelor

Se referă la o metoda ce constă în împărțirea secvențelor în elemente atomice calculând frecvența acestora și determinând următorul element bazându-se pe probabilități pornind de la starea curentă. Cele mai eficiente tehnici ce sunt capabile să învețe probabilități condiționate și sunt adeseori folosite în domeniul machine learning-ului sunt *lanțurile Markov*[15]. Fiecare stare conține multiple stări anterioare prin creare secvențelor *n-grams*[16]. Un articol ce se focusează pe problema creării și antrenării unui model bazat pe *n*-grams, utilizând nivelele originale din „*Super Mario Bros*”, este scris de către Dahlskog ș.a. , el reușind să genereze astfel noi nivele [18]. Procesul de antrenare folosea diferite nivele ale lui *n*, și în timpul generării putem observa că pentru 1-grams avem un nivel generat aproape aleatoriu, în timp ce 2-grams și 3-grams se aproprie mai mult de un nivel real (*Figura 11*).

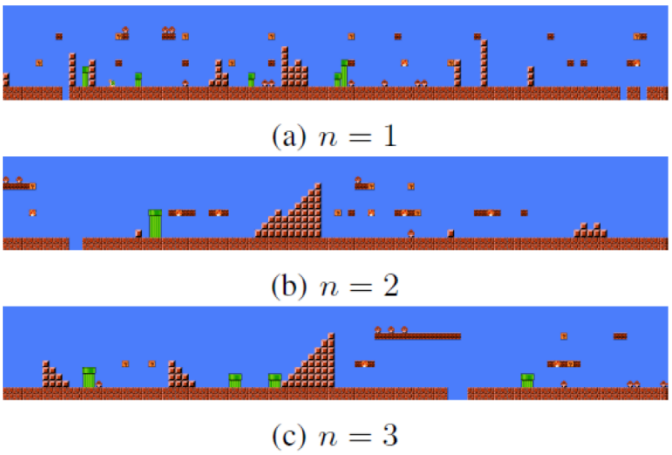


Figura 11 – 1,2,3-grams [18]

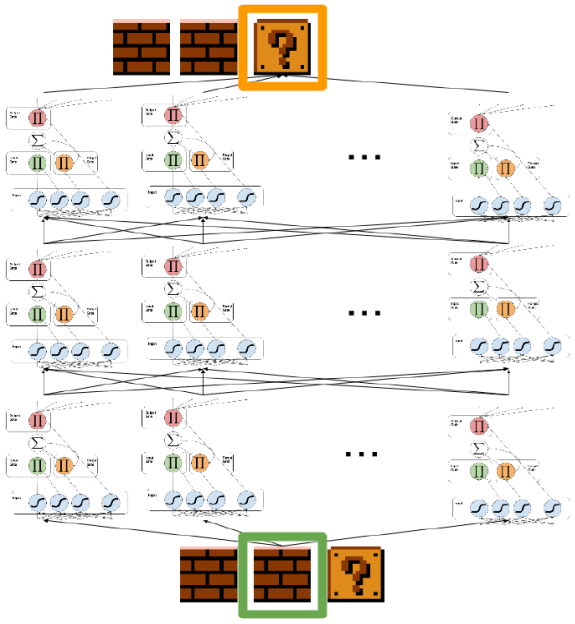


Figura 12 - Modul de generare al următorului element de conținut; verde reprezintă elementul de input și portocaliu cel de output [19]

ii) Propagarea înapoi

Încă de la începuturile PCGML pentru dezvoltatori rețelele neuronale artificiale au reprezentat funcția universală cu ajutorul căreia putem aproxima rezultatele dorite. Metoda cea mai des întâlnită în antrenarea acestor rețele este propagarea înapoi.

[15] Lanțurile Markov reprezintă un proces stochastic ce este caracterizat prin faptul că stările ulterioare ale unui sistem depind de cele anterioare.

[16] Secvență continuă de *N* elemente adeseori formate pe baza unui text.

Astfel o rețea LSTM a fost implementată de către Summerville și Mateas, rețea ce va urma să genereze conținut pentru „*Super Mario Bros”*. După ce modelul a fost antrenat folosind reprezentări vectoriale ale nivelelor, acesta a reușit să genereze nivele noi [19]. În articol harta ce urmează sa fie procesată este reprezentată în 3 moduri secvențiale diferite, cu scopul de a găsi reprezentarea cu cele mai bune rezultate. În cele din urmă au reușit să adauge rețelei noi informații precum adâncimea secvenței relativ la geometria nivelului, astfel reușind să antreneze modelul nu doar pe structuri de bază dar și pe progresia nivelului și unde acesta ar trebui să se termine. În *Figura 12* se poate observa structura internă a procesului de generare.

Alt exemplu distinct de generare de conținut în jocuri folosind rețele LSTM este reprezentat de generarea de cărți pentru „*Magic: The Gathering”*[17]. Modelul creat de Morgan Milewicz, după ce a fost antrenat pe un set întreg de cărți de joc, a reușit să genereze cărți noi [20] . Cărțile au fost reprezentate prin câmpuri de text ale costului, tipului, numelui etc. El a menționat că un dezavantaj al folosirii acestei reprezentări o constituie imposibilitatea de a condiționa conținutul câmpurilor ce apar mai târziu în secvența cărții (*Figura 13*).

Figura 13 - Specificarea parțială a unei cărți urmate de output

3.2. Reprezentare sub formă de grilă

Majoritatea nivelelor din jocuri 2D pot fi reprezentate folosind grile cu două dimensiuni. Câteodată această reprezentare nu se potrivește perfect din cauza formelor sau a alinierii unor elemente de

[17] Joc de cărți ,creat de matematicianul Richard Garfield, ce se joacă în 2 sau mai mulți jucători, jocul reprezentând o bătălie între vrăjitori.

conținut, dar în același timp putem spune că această reprezentare este cea mai naturală pentru multe tipuri de nivele precum temnițe, hărți strategice chiar și jocuri platformă.

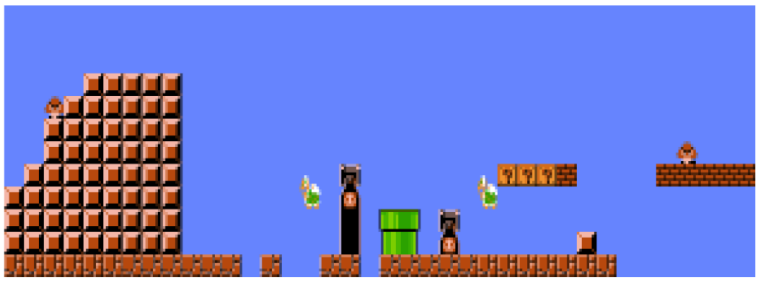


Figura 14 - Nivel din Super Mario Bros generat de lanțuri Markov multidimensionale

A) *Numărarea frecvențelor*

După ce am analizat lanțurile Markov unidimensionale, pasul următor este folosirea lanțurilor Markov multidimensionale sau MdMCs [21]. Diferența între cele două este că cele unidimensionale starea este reprezentată printr-o dimensiune liniară singulară în timp ce în MdMCs starea este reprezentată printr-o întreagă vecinătate, oferind dependințe de la mai multe stări în direcții multiple. Astfel, o nouă abordare a generării de nivele a apărut când Snodgrass și Ontañón au folosit în cercetarea lor MdMCs. Primul pas a fost reprezentarea hărții din *„Super Mario Bros”* sub formă de grilă. Pentru fiecare element din grila rezultată modelul calcula probabilitățile asociate tipului său. De data aceasta modelul lua în considerare întreaga vecinătate și nu doar starea precedentă. Unul dintre rezultatele lor se poate observa în *Figura 14* . După noua lor abordare, au utilizat și câmpuri aleatorii Markov sau MRF. Modelul creat a reușit să se descurce mult mai bine față de standardul MdMC (în generarea nivelelor de *„Kid Icarus”)* [22] .

B) *Propagarea înapoi*

Jan ș.a. au arătat cum autoencoderii sunt capabili să fie antrenați să reproducă nivele din originalul *„Super Mario Bros”*. Autoencoderii au fost antrenați pe porțiuni ce aparțineau nivelelor originale, cu scopul de a le transforma într-o reprezentare mult mai generică. De asemenea ei au descoperit că dimensiunea cea mai potrivită a porțiunilor pe care autoencoderii sunt antrenați este de lungime 4 [23]. Autoencoder-ul poate poate fi folosit mai târziu pentru a diferenția nivelele generate de cele originale și în același timp să repare nivelele ce nu pot fi jucate prin schimbarea caracteristicilor elementelor. Această procedură fiind una dintre cele mai bune alegeri după generarea nivelelor folosind PCG (nu doar în Machine Learning),

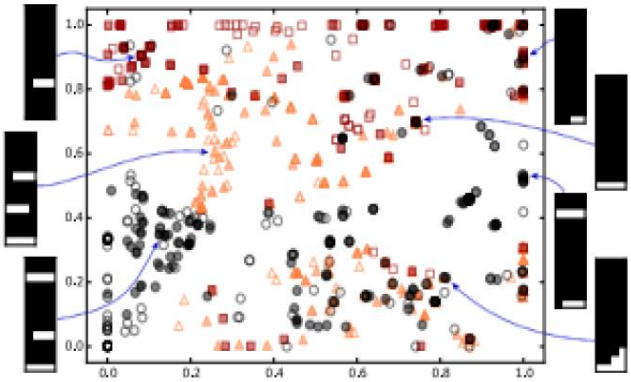


Figura 15- Detectarea nivelului ce conține o anumită porțiune de hartă folosind autoencoderi

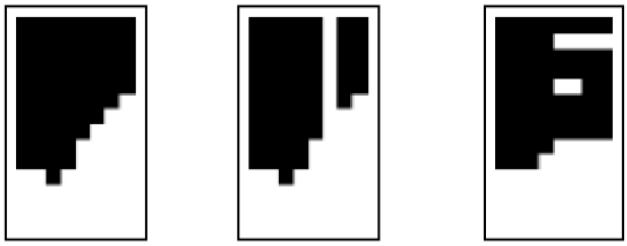


Figura 16 - A) Original B)Generat C) Reparat

deoarece avem posibilitatea de a repara nivele generate greșit. În *Figura 15* se poate observa un exemplu de auto localizare pentru porțiunea curentă fără a fi necesară interacțiunea umană. În *Figura 16* se observă diferența între cele 3 forme de conținut.

O altă întrebuințare a rețelelor neuronale abordată de către Lee ș.a. [24] ce folosesc rețele neuronale convoluționale cu scopul de a prezice locațiile resurselor din cadrul unei hărți de „*StarCraft II*”[18]. De obicei resursele sunt plasate pe hartă în funcție de topologia zonei respective, ele reprezentând un factor esențial în modul în care se desfășoară acțiunea. Folosind acest lucru, eu au transformat harta din joc într-o grilă pe care a introdus-o ca și date de intrare pentru procesul de antrenare al modelului. Un dezavantaj major pentru ei a fost setul de date mic, care de multe ori a dus la *overfitting-ul* [19] rețelei neuronale. Dar prin adăugarea unor pași noi la sfârșitul procesării a reieșit un produs de înaltă calitate permițând dezvoltatorilor să modeleze și să creeze resurse fără a pierde din credibilitatea jucătorilor (*Figura 17*).

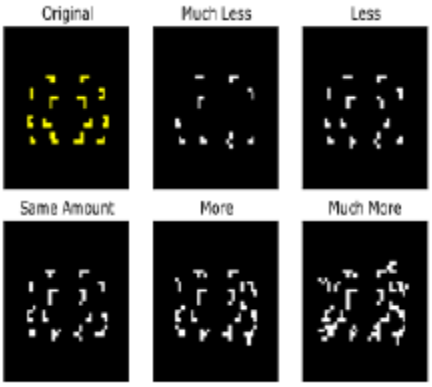


Figura 17 - Aranjarea resurselor pe harta din StarCraft II

C) Factorizarea matricelor

Sunt câteva abordări ce implică folosirea procedeului de factorizare a matricelor pentru a găsi caracteristicile unui nivel. În acest fel caracteristicile sunt obținute prin comprimarea datelor în matrice de mici dimensiuni. Shaker și Abou-Zleikha au reușit să creeze folosind nivele generate prin metode ce nu folosesc machine learning, nivele ce sunt mult mai expresive și mai credibile pentru jucător. Ei au comprimat fiecare nivel în mici matrice bazându-se pe tipul de conținut. Fiecare matrice este apoi factorizată și transformată în *„bucăți de matrice”* și coeficienți, ce reprezintă greutățile specifice fiecărei caracteristici sau model [25].

[18] Joc militar de strategie produs de *Blizzard Entertainment* în 27 Iulie 2010.

[19] Producerea unei analize ce corespunde prea mult unui set de date particular, producând erori în predicții ulterioare datorită incapacității de a modela noi date.

3.3. Reprezentare folosind grafuri

Una dintre cele mai generice metode de a reprezenta datele sunt grafurile, dar generalitatea reprezentării vine cu costul proprietăților structurale (ex. informațiile legate de vecinătatea elementelor fiind oferite implicit prin reprezentarea folosind grile).

i) Maximizarea așteptărilor

Această metodă constă într-un algoritm iterativ ce încearcă să estimeze probabilitatea maximă pentru parametrii unei metode. Această metodă reprezintă o implementare generică pentru orice model ce se bazează pe probabilitatea unui anumit tip de date. Guzdial și Riedl au reușit să folosească clusterizarea ierarhică folosind K-means pentru a antrena modelul. Modelul a fost antrenat cu frame-uri din videoclipuri ce surprindeau oameni în timp ce se jucau *„Super Mario Bros”*, model ce va fi folosit mai târziu pentru a genera noi nivele [26]. Ei au procesat fiecare frame al video-ului folosind OpenCV[20]. Fiecare frame a fost mai târziu combinat pentru a forma structuri geometrice ale nivelului ce vor fi mai târziu folosite ca și date de input pentru generarea întregului nivel. Ei au folosit o structură de graf ce codifica stilurile formelor și probabilitățile relațiilor dintre acestea . Primul pas a fost clusterizarea porțiunilor din nivel cu scopul de a deriva formele proprii ale acestora, după care au clusterizat formele pentru a obține stilurile acestora, iar ultimul constituit calcularea probabilităților relațiilor dintre acestea pentru a determina cum pot fi combinate între ele cu scopul de a genera nivele noi. În *Figura 18* putem observa graful generat pentru câteva porțiuni din nivelurile *Super Mario Bros.*

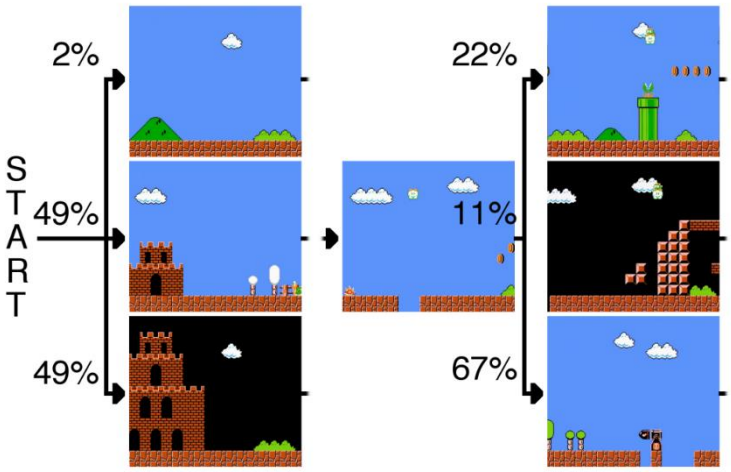


Figura 18 - Graful unui nivel *Super Mario Bros* din nori

[20] Librărie software ce oferă funcționalități in domeniul computer-vision și machine learning.

3.4. Observații și concluzii

În urma analizei desfășurate putem constata că jocurilor de tip platformă sunt cea mai populară țintă pentru generarea procedurală de conținut folosind machine learning. De asemenea, se poate observa faptul că cele mai importante descoperiri sunt pe jocuri ce „hrănesc” conținutul jucătorului în mod secvențial, făcând ușoară analiza completării nivelului.

În concluzie, în cadrul acestui capitol am creat o imagine de ansamblu asupra diferitelor metode de generare a conținutului în jocuri procedural folosind machine learning. Am putut observa cum în zilele noastre PCGML joacă un rol important, dar totodată marea distanță care mai este de parcurs până să se obțină o generare de conținut mai complex.

4. Aplicație

4.1. Specificarea problemei

Conform observațiilor făcute asupra abordărilor populare din cadrul domeniul de PCGML, în cadrul acestui capitol vom propune o implementare ce va încerca să rezolve câteva dintre cele mai importante probleme identificate.

a) Majoritatea jocurilor redau conținutul secvențial jucătorului

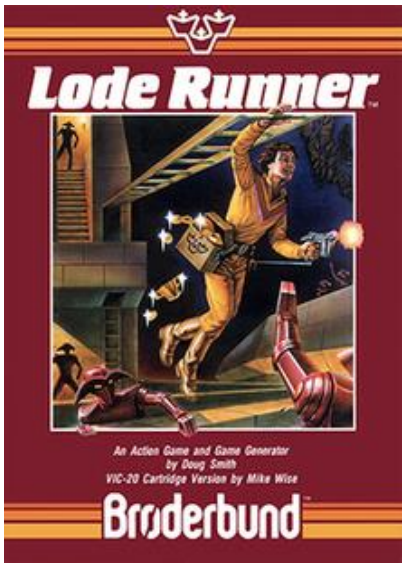
Am dori să tratăm această problemă prin îndepărtarea de la implementări ce au ca și scop crearea de conținut pentru jocuri precum *„Super Mario Bros”,* joc ce este prezent în proporție de 80% în literatură. Jocurile pe care dorim să ne axăm sunt acele jocuri ce oferă jucătorului întregul conținut încă de la început. Având această abordare vom putea analiza ce rată de succes avem în generarea unor altor tipuri de joc decât cele analizate deja. Totodată, nu vom beneficia de aceeași ușurință pentru calculului nivelului de completare a jocului.

b) Diversitate

Una dintre cele mai importate probleme observate în literatură o reprezintă diversitatea. Deși abordări precum rețele LSTM încearcă să rezolve această probleme, ele sunt îngreunate de modul în care anumite jocuri funcționează. Astfel dorim să putem genera conținut ce seamănă foarte puțin cu nivelele de la care am plecat, astfel asigurându-ne ca fiecare nivel va fi interactiv pentru jucător.

Luând în considerare punctele *a)* și *b)* am considerat că cel mai potrivit joc pentru această abordare este ***Lode Runner***. Acesta nu oferă jucătorului conținutul într-un mod secvențial, ci întregul nivel este afișat încă de la început. Un alt avantaj al acestei alegeri este că întreaga arhitectură a nivelului poate fi modificată oricât de mult, astfel dându-ne posibilitatea de a avea un grad de diversitate ridicat.

4.2. Lode Runner



Lode Runner este un joc de tip platformă ce a fost publicat în anul 1983 de către Brøderbund. Față de celelalte jocuri unde focusul era pe acuratețea săriturilor și modul de mișcare, nivelele din *Lode Runner* sunt de tip puzzle .

Aspecte de bază și strategii

Jucătorul este reprezentat printr-o figurină ce trebuie să colecteze toate cuferele de aur și să evite inamicii ce încearcă să-l prindă. După ce a colectat toate cuferele de aur acesta trebuie să ajungă la ultimul etaj al nivelului pentru a putea trece la nivelul următor. Jocul original constă în 150 de nivele ce cresc în dificultatea progresiv astfel oferind o provocare continuă jucătorului. Nivelele oferă diferite structuri cu care jucătorul poate interacționa: scări, cărămizi și bare ce pot fi traversate. De notat este faptul că jucătorul poate săpa gropi de o parte sau de alta a sa. Acest lucru introduce o strategie importantă deoarece pentru a putea săpa o groapă este nevoie de cel puțin 2 spații libere.

Inamici

Gardienii ce încearcă să prindă jucătorul nu au mereu cea mai simplă de prezis traiectorie, alegând drumul cel mai scurt. Ei se mișcă într-o direcție contraintuitivă astfel încât jucătorul trebuie mereu să analizeze și sa aleagă drumul cel mai bun. Acest lucru constituie un factor important deoarece dă posibilitatea jucătorului să găsească metode inovative și strategii potrivite pentru a termina respectivul nivel. Jucătorul are voie să stea pe capul inamicilor fără ca acesta să piardă o viață, astfel dând posibilitatea unor noi strategii precum prinderea inamicilor în capcană în gropile săpate ca mai apoi sa poți traversa peste capul lor pentru ați atinge obiectivul propus.

Alte reguli

De multe ori este nevoie ca ordinea de traversare a nivelul să fie una corectă deoarece alegerea unei direcții diferite poate duce la blocarea sau prinderea jucătorului de către inamici. Totodată, jucătorul trebuie să fi atent la numărul de vieți rămase acestea îi oferă posibilitatea jucătorului de a reîncerca nivelul curent. Jucătorul pornește de la început cu un număr de 5 vieți. Un alt lucru important îl reprezintă timpul, deoarece gropile săpate sunt umplute automat după un anumit timp.

4.3. Dezvoltarea aplicației

Acest capitol va surprinde principalele etape ale dezvoltării unei aplicației. Deși aceste metodologii sunt adesea folosite în crearea software-urilor de sine stătătoare, consider că aplicarea acestor etape duce la o mai ușoară dezvoltare ulterioară a produsului finit.

Analiza

În această fază de dezvoltare vom analiza cerințele sistemului independent de implementare sau proiectare. Aici vom defini problema pe care dorim să o rezolvăm și vom crea o diagramă cu diferitele cazuri de utilizare a aplicației. Tot aici vom dori să analizăm modul în care diferite sisteme comunică între ele și ordinea în care o fac.

Proiectare

În faza de proiectare vom stabili arhitectura sistemului pe baza cerințelor din etapa de analiză. Aici vom configura componentele sistemului precum și comportamentul acestora. Vom analiza totodată și planul de implementare a cerințelor unde vom stabili detalii precum tehnologiile folosite, mediul de dezvoltare, limbajele de programare, structurile de date etc.

Implementare

În cadrul acestei etape vom construi sistemul pe baza cerințelor din etapele anterioare. Aici vom gestiona probleme referitoare la calitatea produsului, biblioteci folosite precum și performanța aplicației.

Testare

Calitatea fiind foarte importantă în cadrul unui produs software, dorim să putem asigura o bună funcționare a sistemului pe o durată lungă de timp. Astfel implementarea și aplicarea testelor pe produsul dorit este necesară, evitând astfel costuri mari pentru modificări ulterioare.

4.3.1. Analiza cerințelor

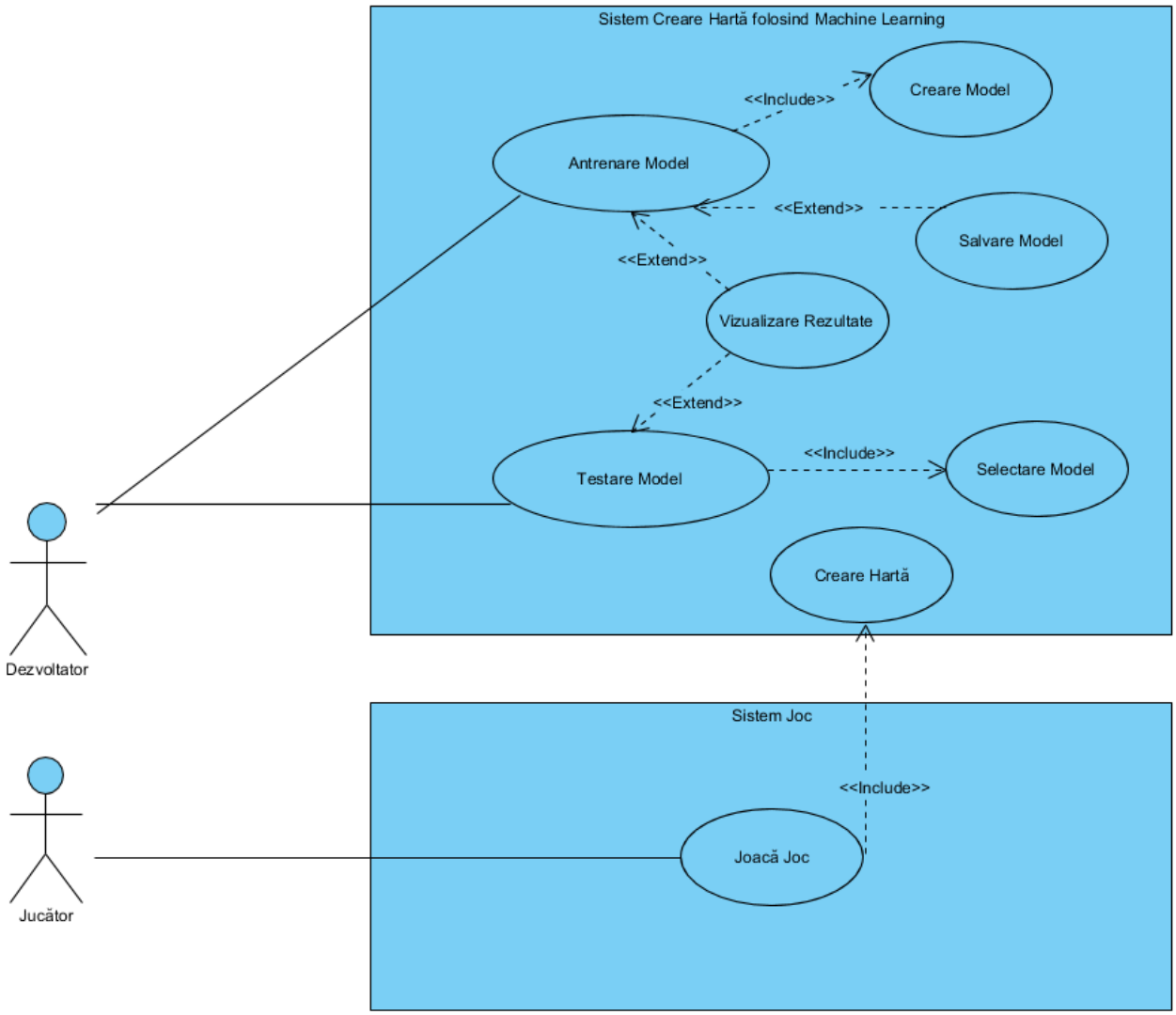


Figura 19 - Diagrama cazurilor de utilizare

Primul pas în analiza cerințelor îl constituie întocmirea diagramei cazurilor de utilizare. După cum se poarte observa *Figura 19* surprinde principalele cazuri în care actorii interacționează cu aplicația.

Antrenarea modelului constituie unul dintre cele mai importante cazuri de utilizare. Dorim ca aplicația să poată crea și antrena un model ce va fi mai târziu folosit în generarea nivelelor pentru joc. De asemenea se poate observa ca se dorește ca după ce modelul a fost antrenat, să se poată analiza rezultatele cu privire la antrenarea sa.

Un alt caz important de utilizare îl constituie posibilitatea testării modelului antrenat de către dezvoltator. Astfel se poate observa cât de bine se descurcă modelul când întâmpină alte date decât cele pe care a fost antrenat. Dorim de asemenea să putem vizualiza rezultatele testării modelului.

Ultimul dar nu cel din urmă caz de utilizare este jucarea jocului de către jucător. Acesta trebuie să poată interacționa cu conținutul pe care noi îl generăm.

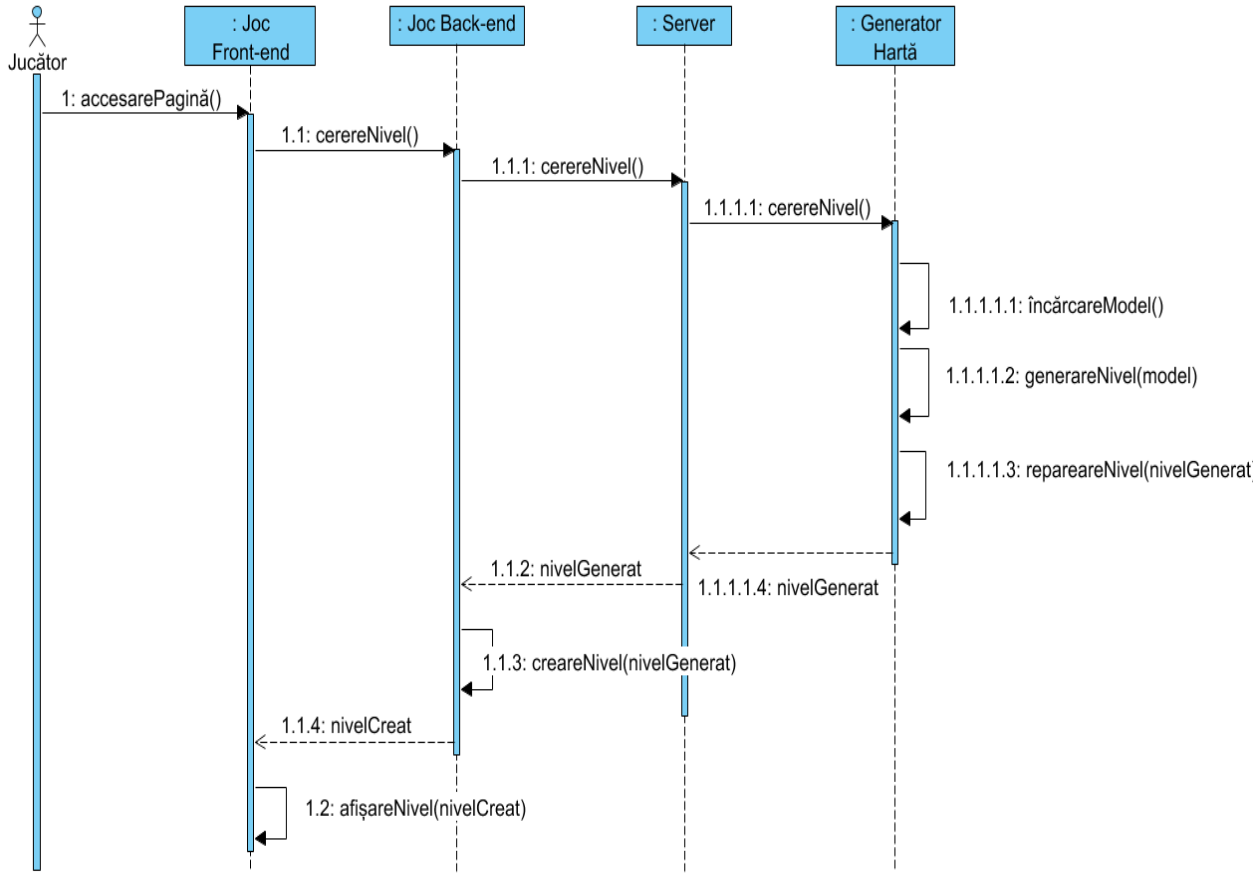


Figura 20 - Diagramă de secvență pentru deschiderea jocului

Pentru a surprinde cazurile de utilizare în amănunt le vom analiza folosind diagrame de secvență specifice. Cu ajutorul acestora vom putea observa modul în care componentele comunică între ele în cadrul aplicației și ordinea în care fiecare sistem trimite și primește mesaje. După cum se poate observa *Figura 20* reprezintă diagrama de secvență pentru momentul în care jucătorul pornește jocul. Deja putem observa că vom avea nevoie de un server cu care trebuie să comunicăm

pentru a primi un nivel generat. Este necesară implementarea unui server deoarece vrem ca diverși jucători să poată comunica cu serviciul implementat pentru generarea nivelului. O altă observație pe care o putem face este aceea că generarea nivelului se face prin încărcarea modelului antrenat și punerea lui în aplicare, dar și nevoia strictă de repararea a nivelului generat.

Ca toate abordările de până acum, vom avea nevoie de o metodă prin care să reparăm nivelul generat, astfel asigurând un nivel ce poate fi jucat. Această reparare se va face în urma procesului de reparare, și va consta în aplicarea unor constrângeri asupra nivelului generat.

Un alt lucru ce se poate observa din *Figura 20* este că avem nevoie ca jocul nostru să poată citi rezultatul primit de la server, pentru a-l putea construi și afișa jucătorului.

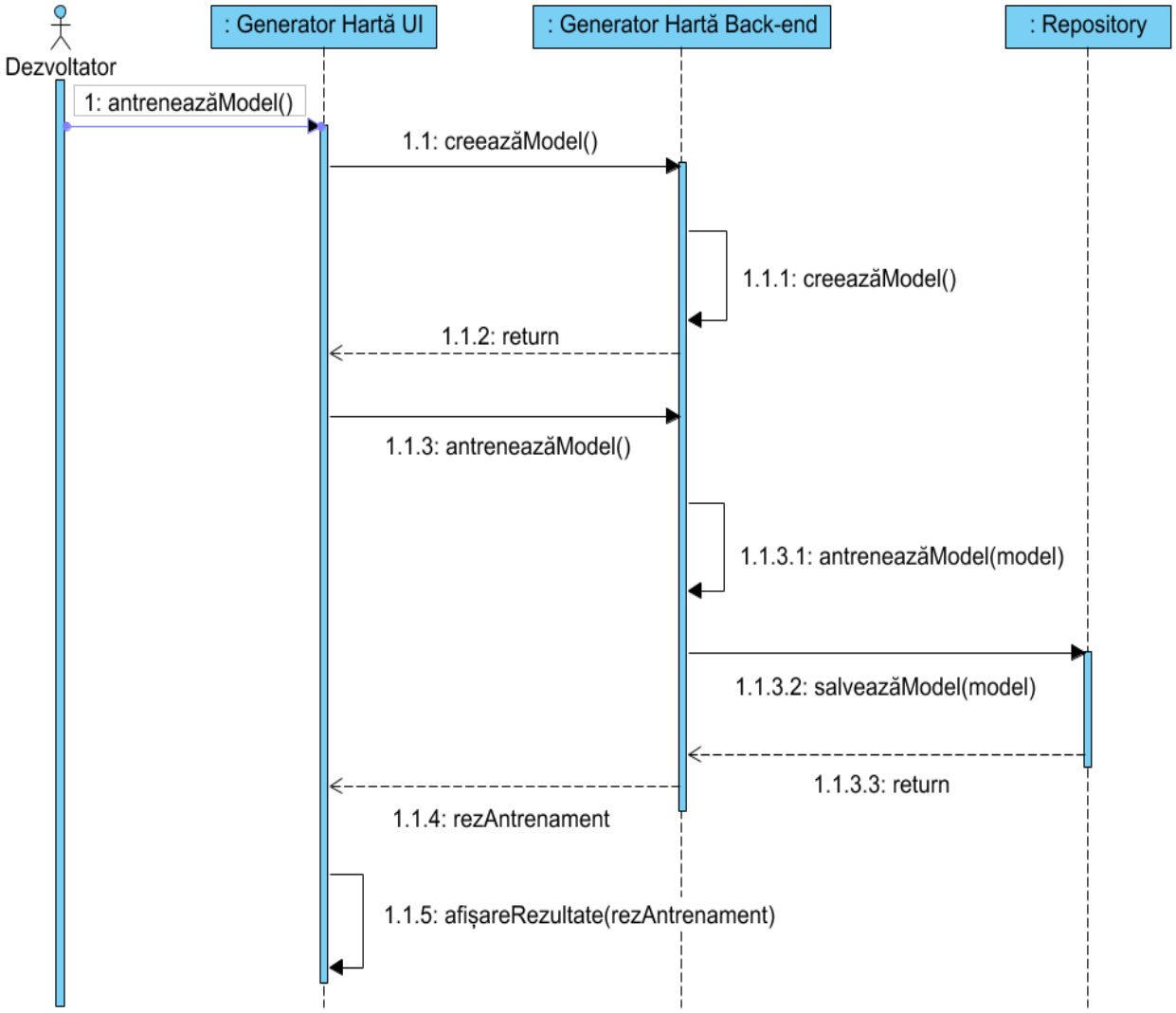


Figura 21 - Diagramă de secvență pentru antrenarea modelului

Figura 21 surprinde o altă diagramă de secvență din cadrul aplicației. Aici vom analiza modul în componente ale aplicației comunică între ele în momentul antrenării unui nivel. Se observă prima etapă din antrenare o constituie creare modelului ce urmează a fi antrenat. După antrenament dorim să se poată salva modelul pentru a putea fi utilizat mai târziu. Această etapă este esențială deoarece pentru generarea de conținut ulterioară vom avea nevoie de modelul pe care l-am antrenat. Tot odată se dorește salvarea modelului la momentul în care acesta este mai bun prin comparație cu cel precedent. Urmând acest proces vom putea crea un model din ce în ce mai performant.

O altă observație importantă referitoare la *Figura 21* o reprezintă posibilitatea de a analiza rezultatele în urma sesiunii de antrenament. Astfel putem observa modul în care modelul evoluează și momentele în care acesta întâmpinăm dificultăți referitor la antrenament. Rezultatele pot fi sub formă de text sau grafice.

4.3.2 Proiectarea aplicației

În urma analizei efectuate în cadrul etapei anterioare, putem realiza arhitectura sistemului pe care dorim să-l implementăm precum și alegerea tehnologii și a limbajelor folosite. Se observă totodată ca vom avea nevoie să implementăm un sistem ce face posibilă comunicarea dintre *Lode Runner* și generatorul de nivele. Acest sistem trebuie să fie rapid ca și durată de răspuns deoarece așteptarea prea îndelungată a jucătorului pentru generarea hărții reprezintă un dezavantaj major.

Scopul acestei aplicații este să analizeze creare de conținut în jocuri folosind machine learning și nu implementarea acestora. Astfel, am ales să folosim un joc deja implementat ce încearcă să imite cu până la cele mai mici detalii jocul original. Implementat în anul 2017 de către Simon Hung, LodeRunner_TotalRecall reprezintă o copie fidelă a faimosului joc *Lode Runner* [27]. Jocul este implementat folosind *CreateJS*[21] având la bază HTML5. Dacă se dorește experimentarea cu această implementare a jocului, acesta poate fi accesat și jucat la următoarea adresă: http://LodeRunnerWebGame.com.

[21] Librărie de funcționalități folosită pentru crearea de conținut interactiv.

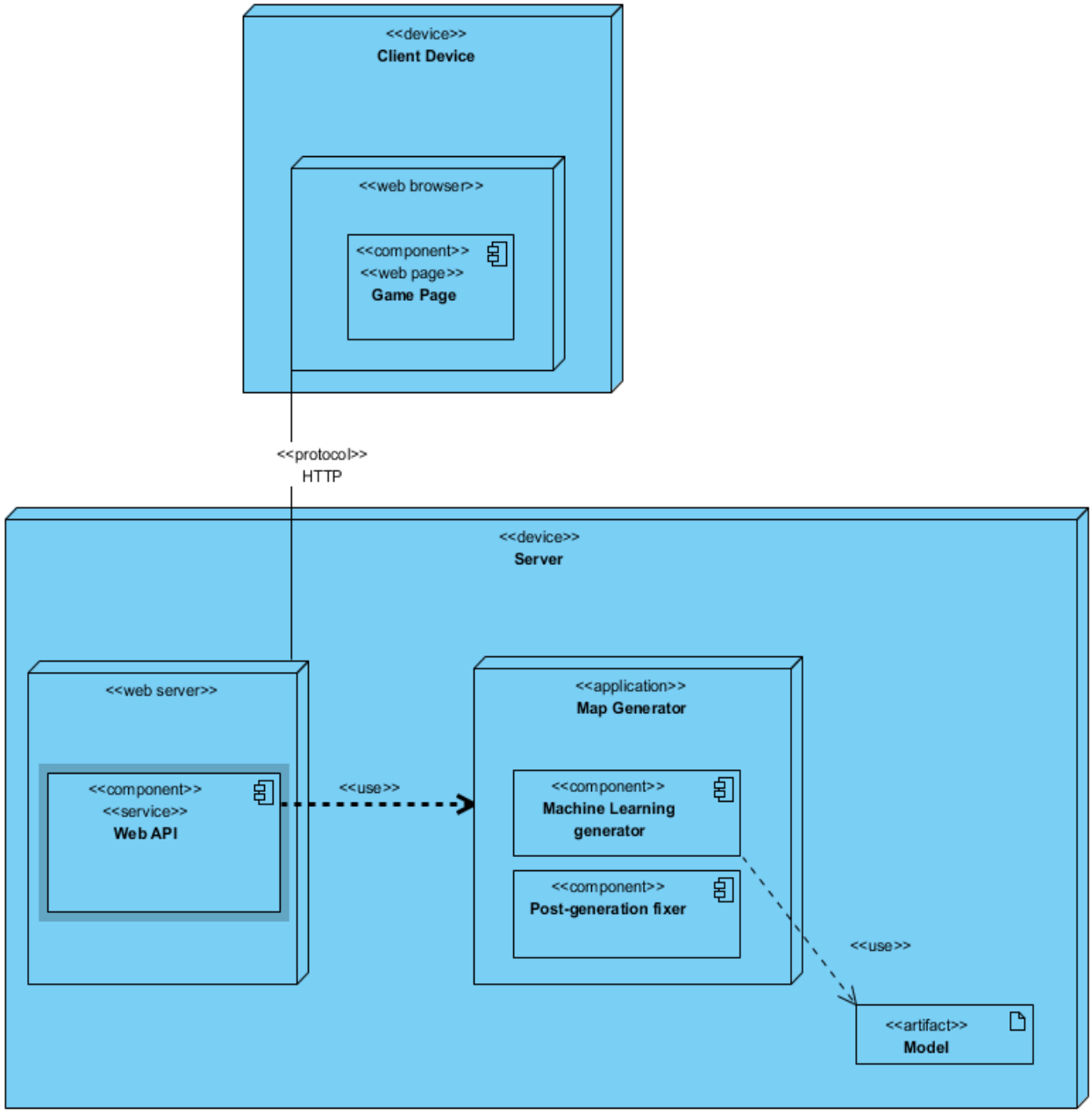


Figura 22 - Diagramă de componente a aplicației

În cadrul *Figura 22* am încercat să surprindem principalele componente ale aplicației și modul în care acestea vor fi structurate. Prima componentă reprezintă dispozitivul de pe care jucătorul se joacă *Lode Runner*. Jocul va fi accesat dintr-o pagină web ce va comunica cu serverul ce are ca și scop generarea nivelului. Comunicare se va face prin HTTP (Hypertext Transfer Protocol) având astfel posibilitatea ca mai mulți clienți să comunice cu serverul nostru [28].

Al doilea nod îl reprezintă serverul ce urmează se fie implementat. Acesta va conține un Web API ce va face posibilă primirea de request-uri de la client și trimiterea răspunsului înapoi. După ce a primit cererea Web API-ul va folosi o componentă de generare a nivelului. Această componentă va consta din 2 părți: o parte ce generează nivelul folosind un model deja antrenat și testat, în timp ce a doua parte are ca și scop repararea nivelului generat, pentru a ne asigura că acesta respectă regulile jocului și ca nu este corupt în vreun fel.

După ce nivelul a fost generat cu succes, acesta va fi trimis către dispozitivul clientului folosind același protocol de comunicare. Avem nevoie ca comunicarea dintre dispozitivul clientului și serverul nostru să fie de tip asincronă deoarece nu dorim ca jocul să se blocheze așteptând după generarea nivelului.

4.3.3 Tehnologii folosite

ASP.NET

Pentru crearea serviciului vom folosi *ASP.NET*. Acesta este o tehnologie Microsoft utilizată pentru crearea de servicii și aplicații web. Arhitectura serviciului va fi una de tip RESTful (Representational State Transfer), deoarece ne oferă flexibilitate în ceea ce privește dezvoltarea serviciului. Cu ajutorul acestuia și cu framework-ul *ASP.NET Web API* vom construi serviciul ce se va ocupa cu transmisia nivelelor între client și serverul construit. Pentru implementarea și dezvoltarea aplicațiilor de tip ASP.NET vom avea nevoi de un mediu specific.

Visual Studio

Acest mediu de dezvoltare integrat (IDE) oferă un pachet întreg de instrumente ce ne permite dezvoltarea aplicaților de tip ASP.NET. Totodată, Visual Studio oferă accesul la tehnologii esențiale ce ușurează procesul de dezvoltare a aplicațiilor web, beneficiind astfel de caracteristicile framework-ului .NET.

Visual Paradigm

Visual Paradigm reprezintă o unealtă UML (Unified Modeling Language) CASE (Computer aided-software engineering) ce oferă un mediu unde putem modela și analiza aplicația ce este sau urmează sa fie implementată. Am folosit această unealtă pentru generarea diagramelor și asigurarea unui model complet și corect al aplicației.

Postman

Cum testarea reprezintă o etapă esențială din procesul dezvoltării, vom avea nevoie de o metodă cu ajutorul căreia să putem testa API-ul creat. Astfel, deoarece testarea în cadrul interfeței cu utilizatorul presupune un nivel ridicat de complexitate, trebuie să putem testa serviciul într-un mod decuplat față de aceasta, în izolare. Postman este construit dintr-un set de unelte ce permite dezvoltatorilor de servicii web să-și testeze aplicațiile. Acesta este folosit pentru a determina daca serviciile expuse de către noi returnează răspunsul corect, în formatul dorit. În felul acesta putem asigura funcționarea corectă a aplicației.

TensorFlow

TensorFlow reprezintă o librărie software ce pune la dispoziția programatorului o multitudine de funcționalități deja implementate din domeniul matematicii. El este adeseori folosit în cadrul domeniului de machine learning, facilitând implementarea pentru tot felul de aplicații precum rețelele neuronale. Oferă programatorilor și inginerilor flexibilitate precum și o metodă rapidă de a pune în aplicare diferite abordări din cadrul acestui domeniu. Noi vom folosi TensorFlow ca și back-end al API-ul ce se va ocupa cu antrenamentul modelului.

Keras

Keras reprezintă o librărie open source scrisă în Python, ce este capabilă să ruleze având ca și parte de back-end TensorFlow, Theano sau Microsoft Cognitive Toolkit. Ea fost dezvoltată pentru a oferi dezvoltatorilor o interfață ce este capabilă să implementeze diverse abordări din domeniul machine learning. Totodată, Keras oferă aplicațiilor extensibilitate și modularitate. Modularitatea librăriei este pusă în evidentă prin posibilitatea configurării modulelor ce pot fi combinate. Componente precum: straturi neuronale, funcții de cost, optimizatori și funcții de activare pot fi combinate pentru a crea noi modele. Un alt motiv important pentru care am considerat Keras este posibilitatea de a ne antrena rețeaua pe placa video, astfel micșorând considerabil timpii de antrenare a modelului.

PyCharm

Deși deja folosim Visual Studio ca și mediu de dezvoltare al Web API-ului, avem nevoie de alt mediu pentru care să ne faciliteze utilizare librăriei Keras și a limbajului de programare Python. PyCharm dezvoltat de către JetBrains ne oferă o gamă largă de unelte pentru dezvoltarea și mentenanța aplicațiilor scrise în Python. Câteva avantaje de care dispunem prin alegerea acestui mediu de dezvoltare sunt: analiza codului, interfață accesibilă de debugging și testare unitară integrată.

Git

Git reprezintă un sistem de control ce se ocupă cu urmărirea schimbărilor din cadrul unei aplicații. Pentru a deține controlul complet asupra versiunilor aplicației vom folosi Git împreună cu SourceTree. Cel din urmă ne va oferi o interfață ușoară de înțeles ce va facilita procesul de mentenanță.

Limbaje de programare: Python și C#

4.3.3. Implementare

Configurarea jocului

Primul pas în implementarea aplicației îl constituie asigurarea bunei funcționări a jocului *Lode Runner*. După ce am clonat LodeRunner_TotalRecall local vom încerca să-l pornim. Deoarece jocul a fost implementat folosind javascript, vom folosi Visual Studio pentru a îl putea porni. În

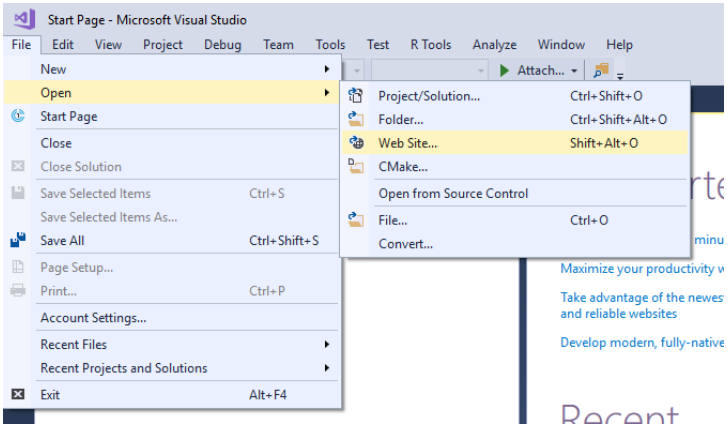


Figura 23 -Pornire joc Lode Runner folosind Visual Studio

Figura 23 putem observa ca Visual Studio ne oferă opțiunea de a porni un proiect ca și un Web Site, exact ceea ce avem nevoie pentru a ne porni jocul. După ce am deschis proiectul putem observa structura jocului. După cum se poate observa putem regăsi diferite fișiere javascript ce se ocupă cu diferite funcționalități ale jocului (*Figura 24*).

Pentru a putea inițializa jocul trebuie să navigăm la fișierul *lodeRunner.html*, iar după ce apăsăm click dreapta pe acesta alegem opțiunea *View in Browser*. Datorită versiunii optimizate de ISS pentru dezvoltatori, ISS Express, ce este integrată în Visual Studio avem posibilitatea de a ne găzdui propria pagină web. Procesul pornește automat, imediat după ce am selectat opțiunea. După acest pas ISS Express alege un port unde poate găzdui aplicația pe *localhost*. După cum se poate observa în *Figura 25* în cazul meu ISS Express mi-a găzduit aplicația la adresa *http://localhost:4245/lodeRunner.html*. După ce am testat jocul și am decis că este complet funcțional am trecut la următoarea etapă a implementării.

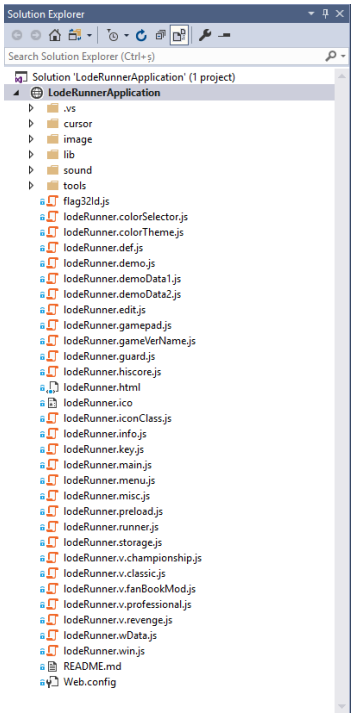


Figura 24 - Structura internă a jocului

Reprezentarea unui nivel

Pentru a putea genera procedural un nivel de *Lode Runner* avem nevoie să identificăm o reprezentare ce poate fi ușor modelată și pe baza căreia să ne antrenăm rețeaua neuronală. Astfel fiecărui element din joc îi va corespunde un caracter în format text.

Figura 25 - Ecranul de start al jocului Lode Runner

Reprezentare grafică	*Tipul elementului*	*Reprezentare text*
	Cărămidă normală	#
	Cărămidă tare	@
	Scară de urcat	H
	Bară de cățărat	-
	Cărămidă falsă	X
	Scară invizibilă	S
	Cufăr de aur	$
	Inamic	0
	Jucător	&
	Spațiu gol	

Tabel 1 - Reprezentare nivel

Conform *Tabel 1* toate elementele jocului sunt reprezentate printr-un caracter text. Spațiul gol din nivel nu va fi reprezentat în niciun fel, astfel evitând complicații ce nu sunt necesare. După cum se observă în *Tabel 1* scara de urcat și scara invizibilă arată la fel dar sunt reprezentate diferit. Aste deoarece ele au funcții diferite pe parcursul jocului. Scara de urcat poate fi folosită pe toată durata nivelului în timp ce scara invizibilă nu este accesibilă decât după ce toate cuferele de au fost colectate de către jucător. O altă diferență importantă o constituie cea dintre cărămida normală și cea falsă, cărămida normală poate fi folosită pe tot parcursul nivelului, dar jucătorul cade prin cărămida falsă.

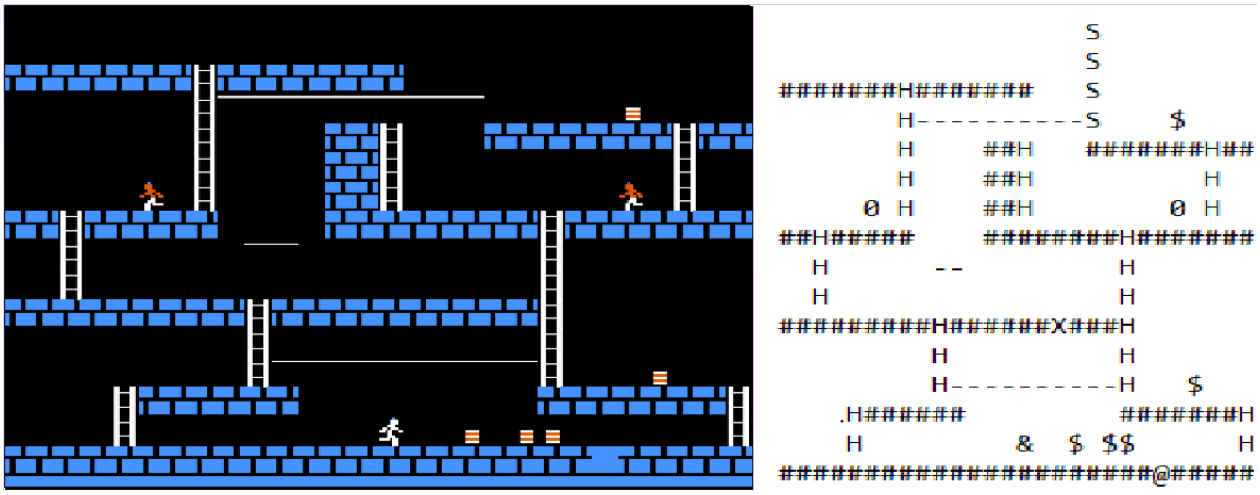


Figura 26 - Nivel Lode Runner și reprezentarea sa

În *Figura 26* putem observa compararea dintre un nivel *Lode Runner* și reprezentarea sa. Datorită acestei reprezentări spațiul ocupat de către fiecare nivel este cu mult mai mic. După ce am stabilit modul în care vom reprezenta nivelul în cadrul aplicației putem trece la următoarea etapă.

Comunicarea între joc și server

Implementarea comunicării între joc și server presupune implementarea Web API-ului ce va gestiona transferul de date dintre cele două. Pentru creare și utilizarea serviciului vom crea un proiect în Visual Studio ce împreună cu jocul vor fi configurate după cum urmează:

- Ne vom asigura că serviciul definește o rută de tip HTTP la adresa: *localhost:port/api/controller*, unde *controller* semnifică numele controlerului implementat.
- Vom implementa un controler cu numele *MapsController()* ce se va ocupa cu tot ce ține de transmiterea nivelelor între jucător și joc.
- Controlerul va implementa o metodă de tip GET ce va răspunde la request-urile din partea jocului. Această metodă se va ocupa și cu apelarea unui scrip python ce va genera și returna un nivel *Lode Runner* (*Figura 28*).
- Vom verifica dacă serviciul poate fi apelat folosind Postman, chiar dacă inițial nu avem ce script rula, vom returna un șir de caractere de test (*Figura 27*).
- Vom implementa un request la Web API folosind AJAX în cadrul jocului.

```
5   namespace LodeRunnerMapsAPI.Controllers
6   {
7       public class MapsController : ApiController
8       {
9           [HttpGet]
10          public IHttpActionResult Level()
11          {
12              var result = Run_cmd("D:/LastYear/Licenta/LodeRunnerMachineLearning/generate/main.py", "");
13              return Ok(result);
14          }
15          public string Run_cmd(string cmd, string args)
16          {
17              ProcessStartInfo start = new ProcessStartInfo
18              {
19                  FileName = "C:/Users/silvi/AppData/Local/conda/conda/envs/LodeRunner/python.exe",
20                  Arguments = string.Format("{0} {1}", cmd, args),
21                  UseShellExecute = false,
22                  RedirectStandardOutput = true
23              };
24              using (Process process = Process.Start(start))
25              {
26                  using (StreamReader reader = process.StandardOutput)
27                  {
28                      string result = reader.ReadToEnd();
29                      return "Test";
30                  }
31              }
32          }
33      }
34  }
```

Figura 28 - Apelarea scriptului python din C#

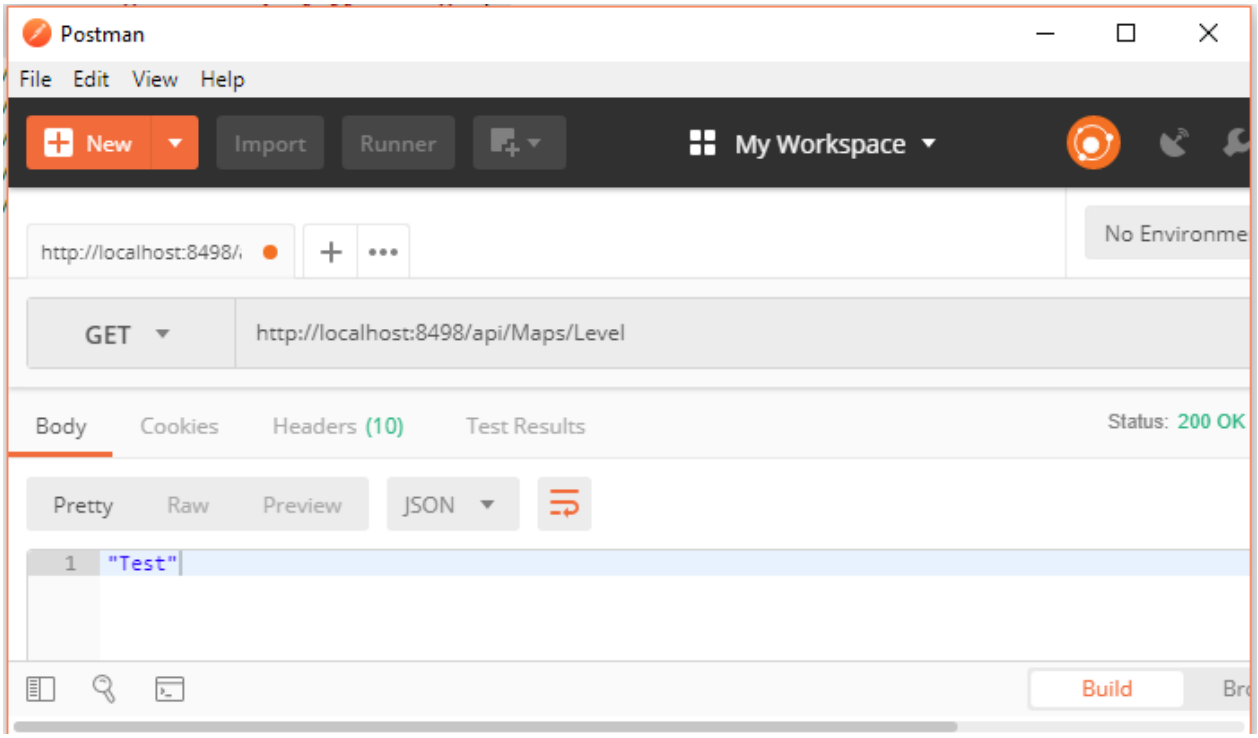


Figura 27 - Testare Web API

Lista figurilor

Bibliografie

[1] N. Shaker, J. Togelius și M. J. Nelson, Procedural Content Generation in Games: A Textbook and an Overview of Current Research, Springer, 2016.

[2] A. M. Smith și M. Mateas, „Answer set programming for procedural content generation: A design space approach," *IEEE Transactions on Computational Intelligence and AI in Games,* vol. 3, pp. 187-200, 2011.

[3] J. Togelius, G. N. Yannakakis, K. O. Stanley și C. Browne, „Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games,* vol. III, pp. 172-186, 2011.

[4] E. J. Aarseth, Cybertext: perspectives on ergodic literature, JHU Press, 1997.

[5] T. Adams, „Dwarf Fortress," *Game [Windows, Mac, Linux], Bay,* vol. 12, 2006.

[6] „Dwarf Fortress Online," [Interactiv]. Available: https://dfsuknfbz46oq.cloudfront.net/p/screenshots/dwarffortress-523e07c2-780b-4831-b95b-3ede852e96e3.png.

[7] S. Samit, „A brief tour of a tiny corner of No Man's Sky," 1 Aprilie 2016. [Interactiv]. Available: https://www.gamesradar.com/no-mans-sky-sheds-light-just-what-youll-do-its-vast-universe/. [Accesat 14 Mai 2018].

[8] X.-J. Mao, C. Shen și Y.-B. Yang, „Image restoration using convolutional auto-encoders with symmetric skip connections. arXiv preprint," *arXiv preprint arXiv:1606.08921,* vol. 2, 2016.

[9] S. J. Russell și P. Norvig, Artificial intelligence: a modern approach, Malaysia; Pearson Education Limited, 2016.

[10] N. J. Nilsson, Introduction to machine learning: An early draft of a proposed textbook, USA; Stanford University, 1996.

[11] I. Goodfellow, Y. Bengio, A. Courville și Y. Bengio, Deep learning, MIT press Cambridge, 2016.

[12] B. Yegnanarayana, Artificial neural networks, PHI Learning Pvt. Ltd., 2009.

[13] R. J. Schalkoff, Artificial neural networks, McGraw-Hill New York, 1997.

[14] L. Medsker și L. Jain, „Recurrent neural networks," *Design and Applications,* vol. 5, 2001.

[15] „RNN," [Interactiv]. Available: http://www.wildml.com/wp-content/uploads/2015/09/rnn.jpg.

[16] F. A. Gers, J. Schmidhuber și F. Cummins, Learning to forget: Continual prediction with LSTM, IET, 1999.

[17] S. M. Bros, *Instruction Booklet,* Nintendo, 1985.

[18] S. Dahlskog, J. Togelius și M. J. Nelson, „Linear levels through n-grams," în *Proceedings of the 18th International Academic MindTrek Conference: Media Business, Management, Content & Services*, 2014.

[19] A. Summerville și M. Mateas, „Super mario as a string: Platformer level generation via lstms," *arXiv preprint arXiv:1603.00930,* 2016.

[20] M. Morgan, „RoboRosewater," 2016. [Interactiv]. Available: https://twitter.com/roborosewater?lang=en.

[21] W. Ching și S. a. N. M. Zhang, „On multi-dimensional Markov chain models," *Pacific Journal of Optimization,* vol. 3, nr. 2, 2007.

[22] S. Snodgrass și S. Ontanon, „Learning to generate video game maps using markov models,” *IEEE Transactions on Computational Intelligence and AI in Games,* vol. 9, nr. 4, pp. 410-422, 2017.

[23] R. Jain, A. Isaksen, C. Holmgard și J. Togelius, „Autoencoders for level generation, repair, and recognition,” în *Proceedings of the ICCC Workshop on Computational Creativity and Games*, 2016.

[24] S. Lee, A. Isaksen, C. Holmgard și J. Togelius, „Predicting Resource Locations in Game Maps Using Deep Convolutional Neural Networks,” în *The Twelfth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. AAAI,* 2016.

[25] N. Shaker și M. Abou-Zleikha, „Alone We Can Do So Little, Together We Can Do So Much: A Combinatorial Approach for Generating Game Content,” *AIIDE,* vol. 14, pp. 1-1, 2014.

[26] M. Guzdial și M. Riedl, „Game level generation from gameplay videos,” în *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[27] S. Hung, „LodeRunner_TotalRecall,” 2017. [Interactiv]. Available: https://github.com/SimonHung/LodeRunner_TotalRecall.git.

[28] R. a. G. J. a. M. J. a. F. H. Fielding, L. Masinter, P. Leach și T. Berners-Lee, Hypertext transfer protocol--HTTP/1.1, 1999.