UNIVERSITATEA TEHNICĂ "GHEORGHE ASACHI" DIN IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

SPECIALIZAREA CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI

LUCRARE DE DIPLOMĂ – RAPORTUL 2

ÎNVĂȚAREA MEDIULUI DINAMIC AL UNUI JOC ORIGINAL FOLOSIND METODE DE ÎNVĂȚARE CU ÎNTĂRIRE IERARHICĂ

Coordonator,
Prof. Florin Leon

Student,
Vlad Batalan

Iași, 2022

Cuprins

1. Proiectarea hardware/software a aplicației

1.1. Arhitectura generală a aplicației

Structura proiectului este alcătuită din două entități care interacționează prin intermediul unui protocol de comunicare. Cele două entități implicate sunt: variantă minimală a jocului original care furnizează un mediu dinamic pentru agent și aplicația responsabilă cu antrenarea, generarea și utilizarea modelelor pentru a completa un nivel al jocului (fig. 1).

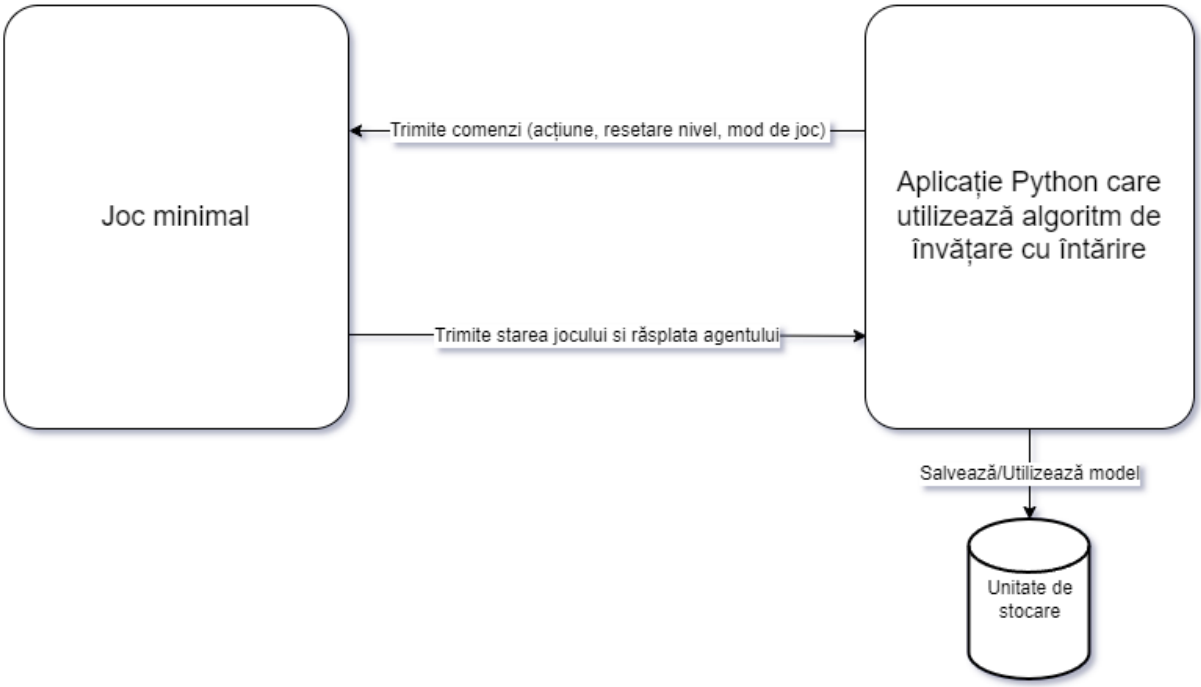

Figura 1 – Arhitectura generală a aplicației

1.2. Structura jocului minimal

Pentru interoperabilitate, a fost realizată o variantă minimală a jocului. Aceasta prezintă câteva sisteme în plus care au ca scop final posibilitatea de a fi controlat de către o aplicație externă. Aceasta va utiliza *GameAPI*-ul pentru a seta modul în care jocul va fi executat (cu interfață grafică sau fără interfață grafică), pentru a controla fazele unui nivel (restart complet, restart parțial) și pentru a realiza o comunicare bidirecțională dintre joc, care oferă starea curentă a nivelului, și aplicațiile externe care pot furniza acțiuni pe care să le execute entitatea Player (fig. 2).

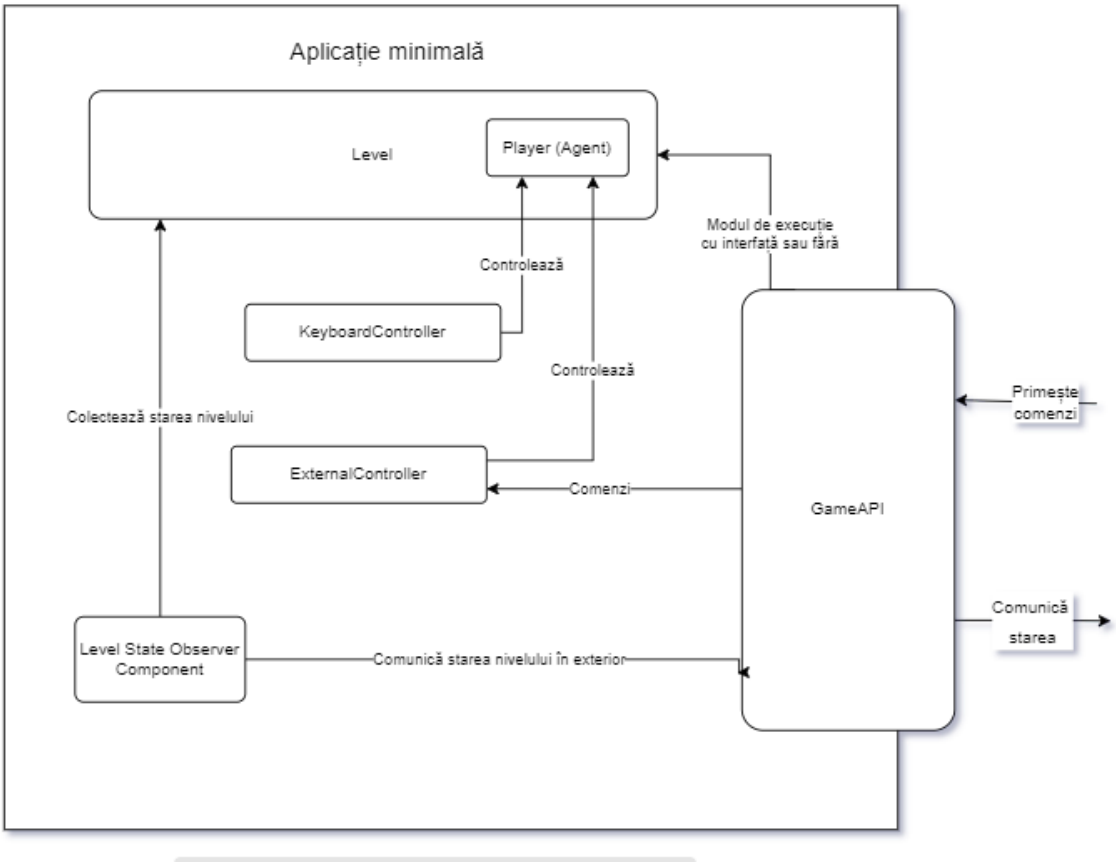

Figura 2 – Arhitectura jocului minimal

Componentele mai importante ale aplicației minimale sunt:

- Level State Observer component – reprezintă unitatea responsabilă cu monitorizarea stării nivelului; aceasta transformă date despre nivel precum hartă, pozițiile entităților, starea obiectelor, starea nivelului (câștigat, pierdut, început) într-un format xml sau JSON și transmite API-ului.
- GameAPI – este componenta care comunica direct cu exteriorul; aceasta primește comenzi într-un anumit format și decide entitatea care să le execute.
- ExternalController – este un controller special creat pentru primirea comenzilor de la aplicații externe și manifestarea acestora asupra agentului (Player).

1.3. Structura aplicației responsabile cu procesul de învățare cu întărire

În stadiul curent, această parte nu a fost încă implementată. Componentele de care are nevoie sunt: un API prin intermediul căruia poate configura și comunica cu o instanță a jocului,

un interpretor de stare pentru trecerea din format xml sau JSON al unei stări în date ce pot fi ușor utilizate pentru procesul de învățare, o unitate de stocare de unde pot fi accesate sau salvate modele, o componentă care gestionează diferiții algoritmi utilizați (QLearning, Hierarchical QLearning, etc.) și componenta principală (*MainComponent*) responsabilă cu generarea de comenzi pentru aplicația tip joc (fig. 3)

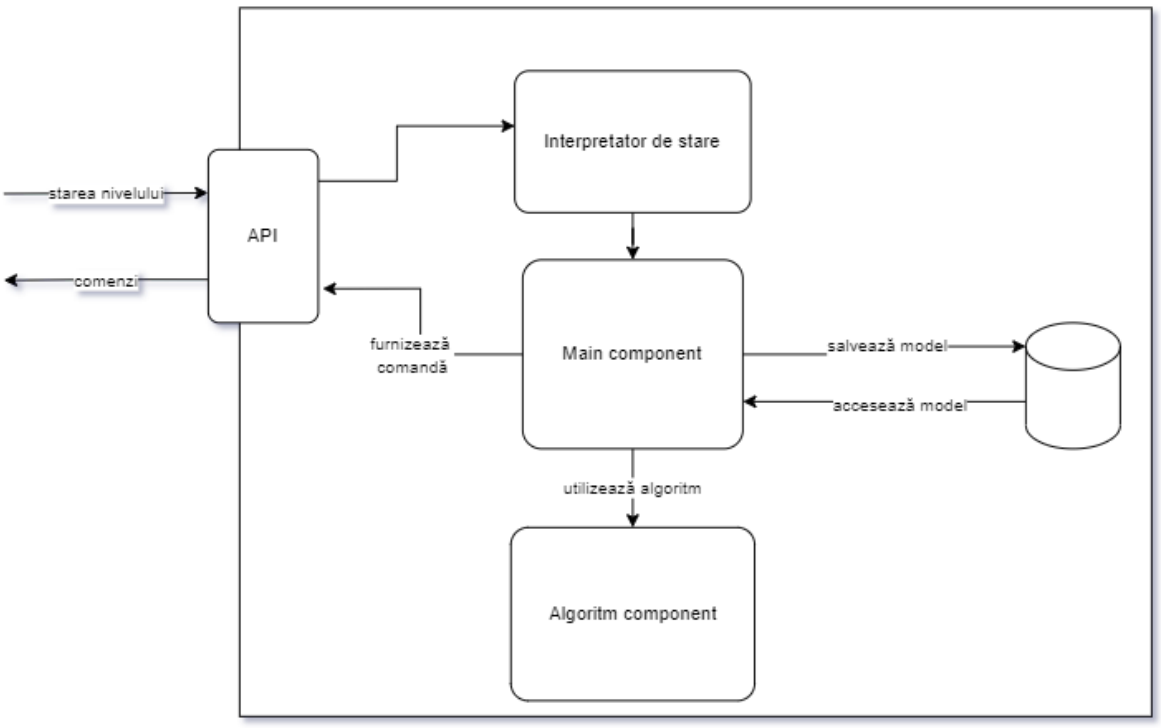

Figura 3 – Structura generală a componentei de învățare cu întărire

2. Rezultate intermediare obținute

În stadiul curent al dezvoltării proiectului, un rezultat intermediar notabil este implementarea unei variante minimale a jocului care poate să interacționeze cu modulul responsabil de procesul de învățare cu întărire.

Aplicația are nevoie de o posibilitate de a comuta între varianta de a utiliza o interfață grafică sau de a se executa fără aceasta, facilitând astfel reducerea timpului necesar antrenării. În modul cu interfață grafică, un cadru de joc, considerat unitatea atomică de luare de decizii pentru algoritmul de învățare cu întărire, este redat odată la 16.66 milisecunde, spre deosebire de modul de execuție fără interfață în care momentul când se trece la următorul cadru poate fi programat de către modulul de învățare. De asemenea, eliminarea tranzițiilor de la începutul și finalul unui

nivel, a textelor, a obiectelor care nu sunt esențiale și utilizarea unui sistem de log-uri pentru cazul execuției fără interfață simplifică mediul de lucru (fig. 4).

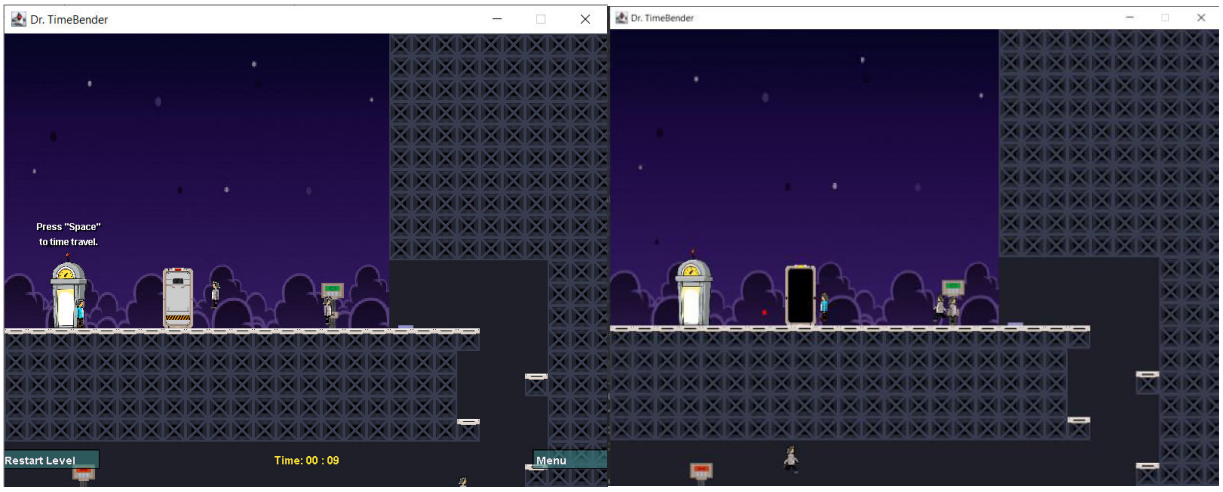

Figura 4 – Varianta veche a jocului / Varianta minimală a jocului

Implementarea unui nou controller pentru gestionarea mișcărilor agentului capabil să primească comenzi direct de la modul de învățare cu întărire este necesară. Acesta ascultă evenimente de la modulul python și le manifestă în mediul de joc. De exemplu, pentru mișcarea jucătorului pe hartă sunt implementate trei tipuri de controllere: *KeyboardController*, care ascultă evenimentele apărute de la tastatură și creează comenzi în funcție de natura acestora, și *ExternalController*, care ascultă evenimente de la surse externe, precum modulul python sau alți clienți (nu este încă implementat) și *FixedController*, utilizat de instanțele din trecut ale jucătorului, care prezintă o listă de comenzi preconcepute, menite să fie executate la un moment specific de timp (fig. 5).

Realizarea unei componente care poate observa starea jocului într-un anumit cadru (*LevelStateObserver*) și transpunerea acesteia într-un format (xml, JSON, etc.) care poate fi ușor transmis la modulul de învățare este esențială. Pentru starea jocului se pot considera diferite aspecte de interes: starea jocului (câștigat, pierdut, început), forma hărții, pozițiile personajelor și stările obiectelor de pe hartă.

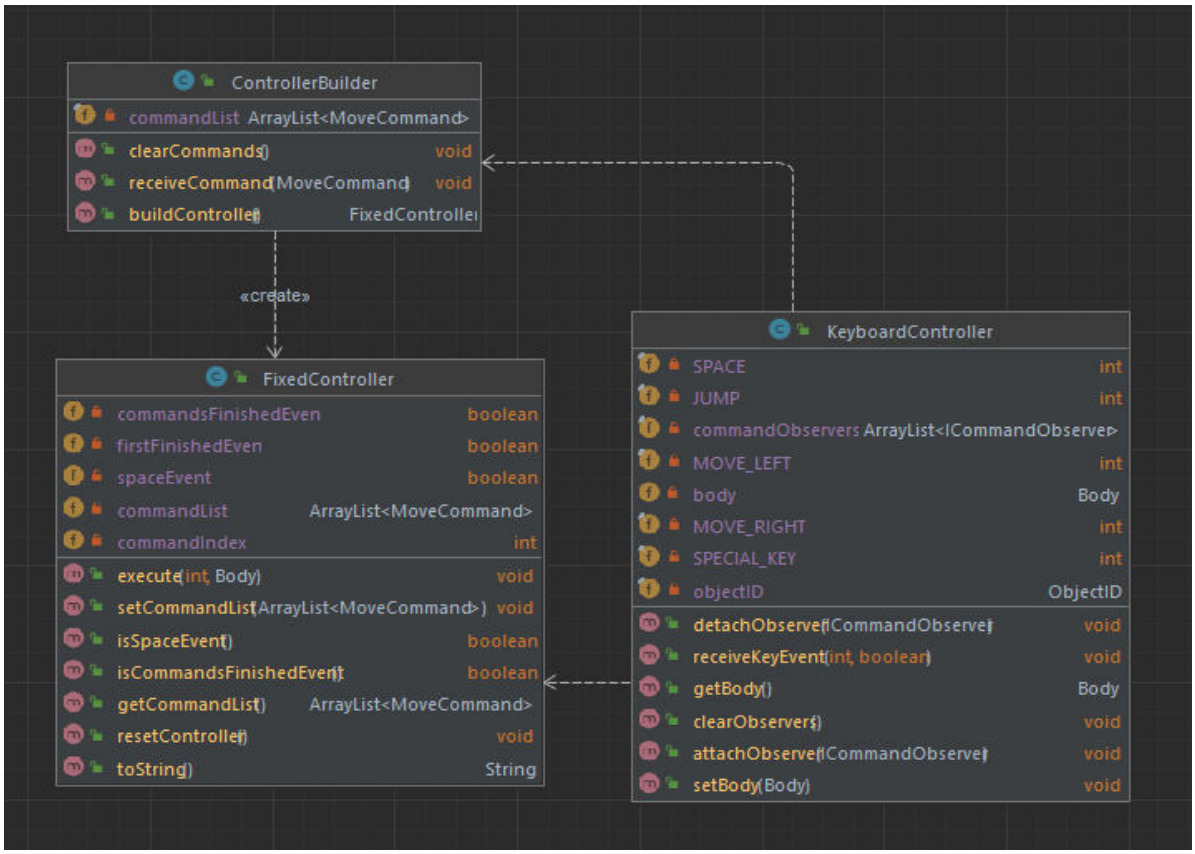

Figura 5 – UML diferite tipuri de controllere implementate

3. Dificultăți/provocări și soluții de rezolvare

3.1. Probleme generate de vechea implementare a sistemului mișcării entităților

Varianta completă a jocului avea implementat un sistem diferit pentru gestionarea mișcării corpului unui obiect în funcțiile de comenzile de mișcare furnizate. Acesta era bazat pe un sistem de steaguri (*MoveLeftFlag*, *MoveRightFlag*, *JumpFlag*) care se activează/dezactivează în funcție de input. O comandă de mișcare era creată direct de către obiectul responsabil de primirea evenimentelor de la tastatura și nu exista niciun controller intermediar. Astfel, această abordare nu acceptă decât controlul playerului doar prin tastatura.

Instanțele din trecut ale unui player sunt entități care urmează aceleași comenzi pe care le-a executat playerul într-o iterație din trecut a fazei de joc. Pentru stocarea acestor comenzi, se utiliza un *ControllerBuider* care urmărea evenimentele de la tastatura si genera o listă de comenzi. Deoarece fiecare comandă avea un cod executat pentru momentul începerii ei și un cod pentru momentul încheierii acesteia (fig. 6), exista următorul șir de evenimente care provoca probleme la mișcarea entităților: Left pressed, Right pressed, Left released, Right released. Acest

bug poate influența drastic procesul de învățare, deoarece înșiruirea provoca pierderea nivelului fără ca utilizatorul să aibă o legătura logică cu aceasta.

Soluția acestei probleme a constat într-un nou sistem de mișcare implementat pentru varianta minimalistă a jocului care este bazat pe un automat finit de stări. Comenzile devin evenimente ce pot provoca treceri ale automatului finit dintr-o stare de mișcare în altă stare de mișcare. Aceasta abordare simplifică inputul pe care trebuie să îl primească un corp pentru a se mișca și rezolvă problema generată de secvența problematică de evenimente (fig. 7).

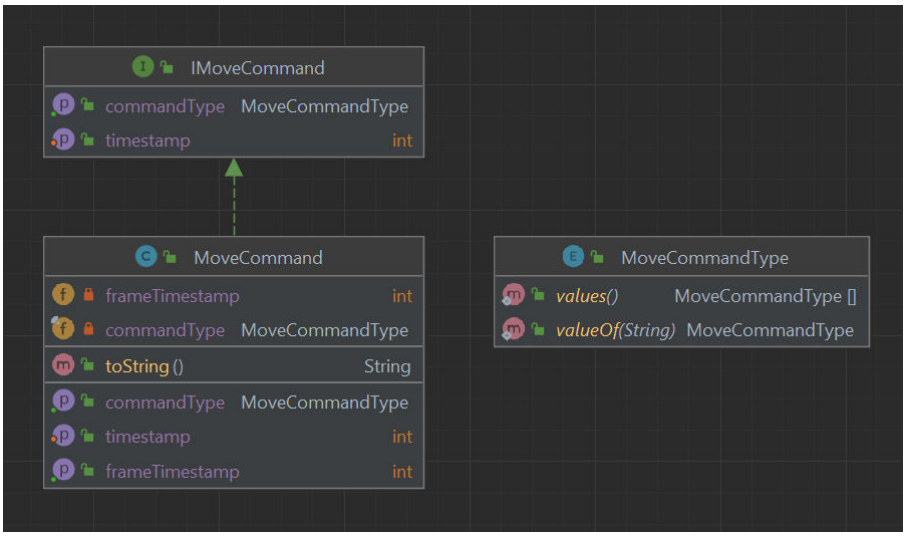

Figura 7 – Noua structură a tuturor comenzilor

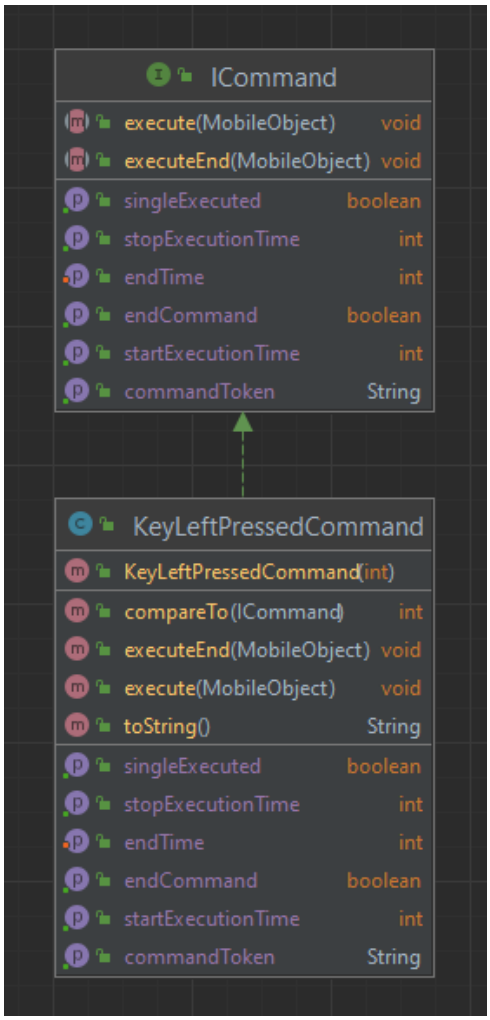

Figura 6 – Structura veche a unei comenzi

Starea de mișcare a unui corp este redată prin compunerea a două automate finite diferite: unul pentru direcția de mișcare (fig. 9) și unul pentru statusul săriturii (fig. 10). Fiecare stare a corpului are o metodă prin care, în funcție de evenimentele primite sub formă de *MoveCommands,* decide următoarea stare în care să treacă, respectând graful de fluență și câteva condiții specifice (fig. 8).

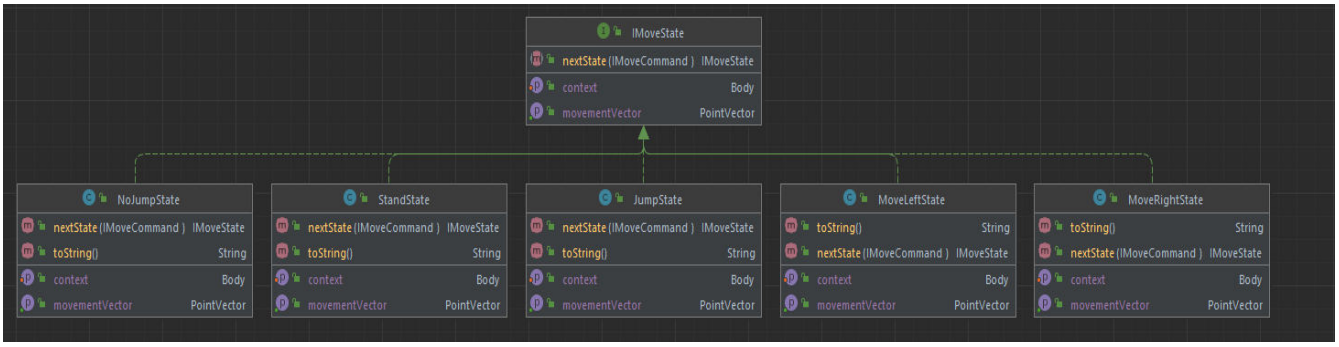

Figura 8 – UML cu stările unui corp

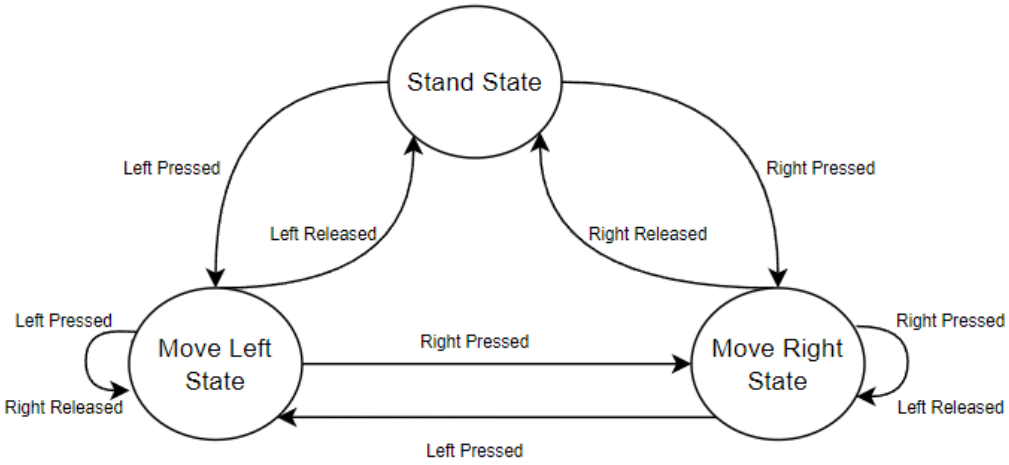

Figura 9 – Automat finit pentru starea de mișcare a unui corp

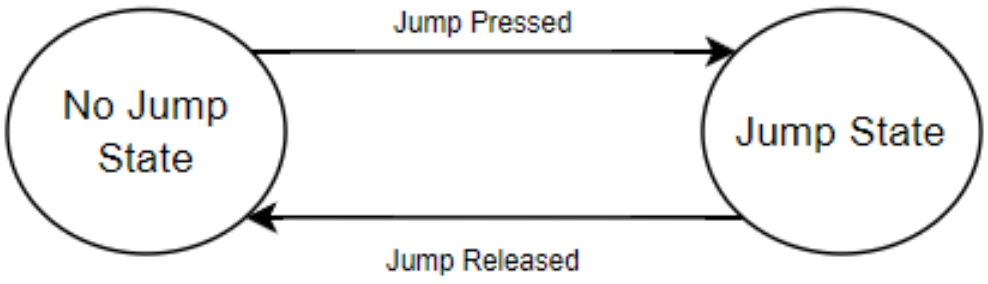

Figura 10 – Automat finit pentru starea de săritură a unui corp