Modelowanie i analiza systemów informatycznych (Informatyka st. II sem. 2)

Dokumentacja projektu nr 3

Dawid Bitner, Daniel Broczkowski, Damian Kwaśniok

18 grudnia 2021

Część I

Opis programu

Celem projektu była implementacja systemu wieloagentowego z użyciem sieci neuronowych/logiki rozmytej lub innych obszarów AI. System ten został użyty do implementacji gry arcade z filmu Tron. Założono, że obiekty będą się poruszać na płaszczyźnie odpowiadającej układowi współrzędnych, a poszczególne ruchy są uzależnione od pozycji gracza i agenta.

Instrukcja obsługi

0.1 Instrukcja wdrożenia

Do poprawnego uruchomienia aplikacji konieczne jest zainstalowanie Unity w wersji *2020.3.22f1*. Korzystając z kontrolera wersji Unity Hub należy utworzyć projekt wskazując lokalizację folderu na dysku w którym znajdują się pliki projektu.

Podział prac

Dawid Bitner - dokumentacja
Daniel Broczkowski - dokumentacja
Damian Kwaśniok - dokumentacja.

Część II

0.2 Model systemu wieloagentowego

Model systemu wieloagentowego to system, który działa samodzielnie w otwartym, rozproszonym środowisku i rozwiązuje pewien problem lub wykonuje określone zadanie. Agent postrzega swoje środowisko i może na nie oddziaływać oraz cechuje się autonomicznością (może działać bez udziału człowieka lub innego agenta).
System wieloagentowy to sieć luźno powiązanych agentów, którzy współdziałają, aby rozwiązać problemy leżące poza ich indywidualnymi zdolnościami i wiedzą.

Agent:

$$A = \{'up', 'down', 'left', 'right'\}$$

$$S = \{'empty', 'wall'\}$$

$$\begin{gathered} A \times S = \{'up' \times 'empty' = 'wall', \\ \text{'down'} \times 'empty' = 'wall', \\ \text{'left'} \times 'empty' = 'wall', \\ \text{'right'} \times 'empty' = 'wall', \\ \text{'up'} \times 'wall' = 'game\ over', \\ \text{'down'} \times 'wall' = 'game\ over', \\ \text{'left'} \times 'wall' = 'game\ over', \\ \text{'right'} \times 'wall' = 'game\ over'\} \end{gathered}$$

Wieża:

$$A = \{'oczekuj', 'reaktywuj', 'pomoc'\}$$

$$S = \{'blisko', 'daleko'\}$$

$$\begin{gathered} A \times S = \{'oczekuj' \times 'blisko' = 'uciekaj', \\ \text{'oczekuj'} \times 'daleko' = 'ruch', \\ \text{'reaktywuj'} \times 'blisko' = 'pomoc', \\ \text{'reaktywuj'} \times 'daleko' = 'czekaj', \\ \text{'pomoc'} \times 'blisko' = 'ruch', \\ \text{'pomoc'} \times 'daleko' = 'czekaj'\} \end{gathered}$$

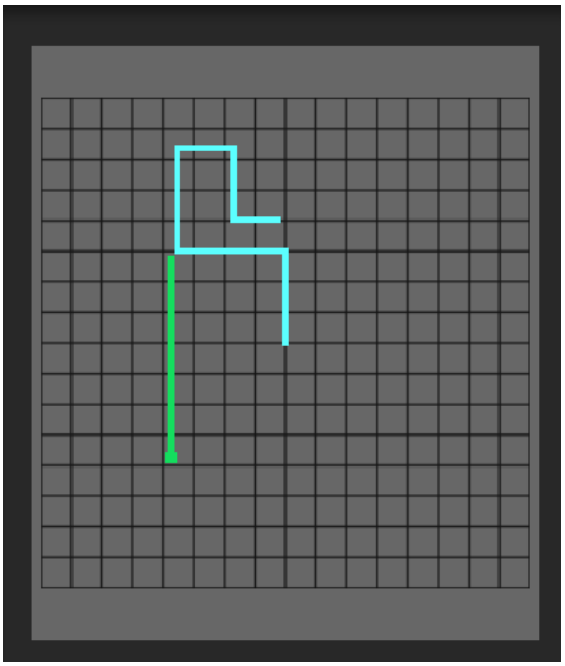
0.2.1 Model matematyczny agenta

Ruch agenta jest uzależniony od jego odległości od gracza. Odległość jest oblicza ze wzoru na odległość między punktami w układzie współrzędnych:

$$|AB| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Komunikacja między graczem, a agentem (przeciwnikiem) odbywa się, gdy odległość między nimi jest większa niż 5. Wtedy jest udzielana wskazówka w którym kierunku ma się udać agent. Pomoc ta odbywa się na pierwszych 100 klatkach gry.

Opis gry



Gra arcade z filmu Tron używa strategii zmuszania przeciwnika do wbiegnięcia na ścianę lub ścieżkę gracza. Algorytm oblicza, który ruch przeciwnika jest najkorzystniejszy i ruch w takim kierunku jest wykonywany.

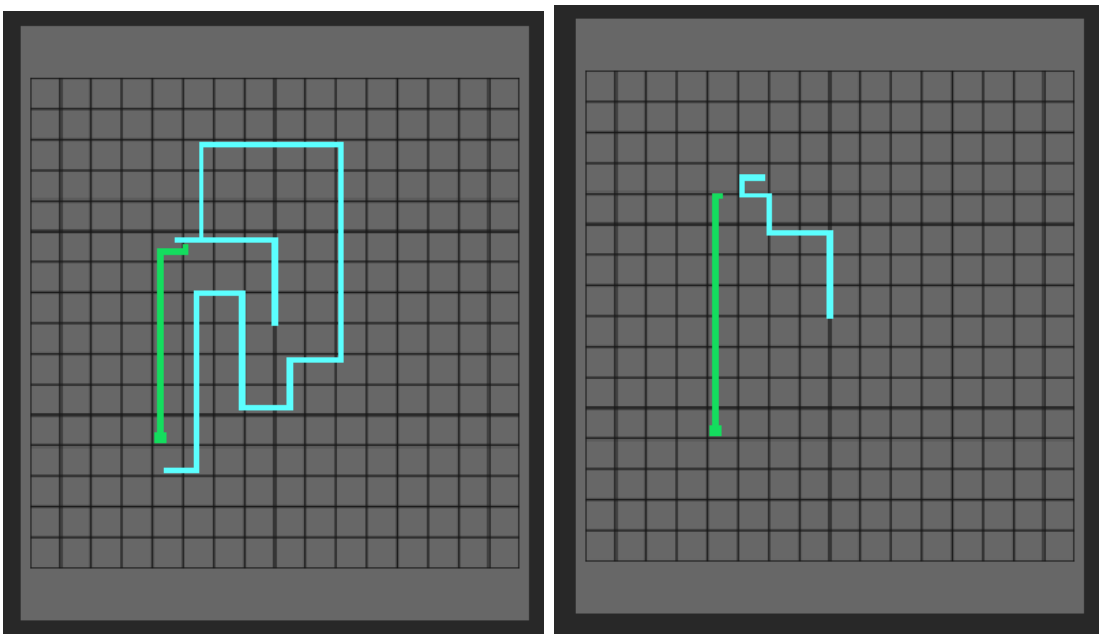
Zalety i wady systemów wieloagentowych

Do głównych zalet systemów wieloagentowych można zaliczyć:
- brak ograniczenia co do platformy na której jest użyty system (możliwość komunikacji sieciowej na urządzeniach mobilnych)
- samouczenie się agentów, pozyskiwanie informacji w procesie komunikacji z innymi agentami
- autonomiczność agentów
- niskie wymagania co do kanałów transmisyjnych
- szybkość działania
- wykorzystanie większego zakresu wiedzy przy podejmowaniu ostatecznej decyzji (wiedza wielu agentów)
- panowanie nad dużą ilością danych

Naszym zdaniem do głównych wad systemów wieloagentowych należą m.in.:
- bezpieczeństwo agentów (agentem może być intruz, koń trojański)
- wdrożenie systemu jest kosztowne, wymaga znacznych nakładów finansowych
- brak sprawdzonych, komercyjnych rozwiązań wykorzystujących system wieloagentowy

Testy

Program był testowany manualnie. Testy wykazały, że gra nie działa do końca poprawnie co jest widoczne na poniższych zrzutach ekranu:



W pierwszym przypadku po wykryciu kolizji program umowożliwiał dalszy ruch gracza. Natomiast w drugim przypadku system wykrył kolizję mimo iż nie nastąpiła. Jest to związane z konfiguracją Colliderów.

Pełen kod programu

0.2.2 agentTower.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class agentTower : MonoBehaviour
{
  public GameObject agent1;
  public GameObject player;
  int reactivation=100;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
    if(reactivation<=0) {
         if(Vector3.Distance(agent1.transform.position,player.transform.
            position)<5)
    {
      Debug.Log("ok");
    }
    else {
      //udziel pomocy
      player.GetComponent<AIAgent>().Communication("left");
    }
    }
    if(reactivation>0) reactivation--;
    }
}
```

0.2.3 AIAgent.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class AIAgent : MonoBehaviour
{
  public float speed=16;

  Vector2 lastWallEnd;


  private string actualDirection="";
  Collider2D wall;
  public GameObject player;
```

```
15  public GameObject wallPrefab;
16  public void Communication(string info){
17    if(counter%1000==0&&info!=actualDirection){
18      ChangeDirection(info);
19    }
20  }
21    // Start is called before the first frame update
22    void Start()
23    {
24        GetComponent<Rigidbody2D>().velocity=Vector2.up*speed;
25    spawnWall();
26    }
27  private int counter=0;
28  float distanceMetric(Vector2 pos)
29  {
30    return Mathf.Sqrt(
31    Mathf.Pow(pos.x-player.transform.position.x,2)
32    +
33    Mathf.Pow(pos.y-player.transform.position.y,2));
34  }
35    // Update is called once per frame
36    void Update()
37    {
38    if(counter%10==0){
39        //oblicz odleglosc od gracza w zaleznosci od kierunku
40      float[] dir=new float[4]{0,0,0,0}; //up,down,left,right
41
42      dir[0]=distanceMetric(new Vector2(transform.position.x,
43      transform.position.y+1));
44      dir[1]=distanceMetric(new Vector2(transform.position.x,
45      transform.position.y-1));
46      dir[2]=distanceMetric(new Vector2(transform.position.x-1,
47      transform.position.y));
48      dir[3]=distanceMetric(new Vector2(transform.position.x+1,
49      transform.position.y));
50      //wybierz kierunek o najmniejszym koszcie
51      int index = 0;
52      for (int i = 0; i < dir.Length; i++)
53      {
54        if (dir[i] < dir[index]) { index = i; }
55      }
56      //zmien kierunek
57      if(index==0) ChangeDirection("up");
58      if(index==1) ChangeDirection("down");
59      if(index==2) ChangeDirection("left");
60      if(index==3) ChangeDirection("right");
61    }
62    counter++;
63    fitColliderBetween(wall,lastWallEnd,transform.position);
64    }
65  void ChangeDirection(string direction)
66  {
67    if(actualDirection==direction) return;
68      actualDirection=direction;
69      if(direction == "up"){
```

```
70          GetComponent<Rigidbody2D>().velocity=Vector2.up*speed;
71          spawnWall();
72        }
73        if(direction == "down"){
74          GetComponent<Rigidbody2D>().velocity=-Vector2.up*speed;
75          spawnWall();
76        }
77        if(direction == "left"){
78          GetComponent<Rigidbody2D>().velocity=-Vector2.right*speed;
79          spawnWall();
80        }
81        if(direction == "right"){
82          GetComponent<Rigidbody2D>().velocity=Vector2.right*speed;
83          spawnWall();
84        }
85    }
86
87    void spawnWall(){
88      lastWallEnd=transform.position;
89
90      GameObject c= Instantiate(wallPrefab,transform.position,Quaternion.
            identity);
91      wall=c.GetComponent<Collider2D>();
92    }
93
94    void fitColliderBetween(Collider2D what, Vector2 a, Vector2 b){
95      what.transform.position=a+(b-a)*0.5f;
96      float distance = Vector2.Distance(a,b);
97      if(a.x != b.x){
98        what.transform.localScale=new Vector2(distance+1,1);
99      }
100     else {
101       what.transform.localScale=new Vector2(1,distance+1);
102     }
103
104   }
105
106   void OnTriggerEnter2D(Collider2D co) {
107     if (co != wall) {
108         print("Player lost: " + name);
109         Destroy(gameObject);
110     }
111 }
112 }
```

0.2.4 move.cs

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class move : MonoBehaviour
6 {
```

```
7   public KeyCode upKey;
8   public KeyCode downKey;
9   public KeyCode leftKey;
10  public KeyCode rightKey;
11
12  public float speed=16;
13
14  Vector2 lastWallEnd;
15
16  Collider2D wall;
17
18  public GameObject wallPrefab;
19
20    // Start is called before the first frame update
21    void Start()
22    {
23        GetComponent<Rigidbody2D>().velocity=Vector2.up*speed;
24    spawnWall();
25    }
26
27    // Update is called once per frame
28    void Update()
29    {
30        if(Input.GetKeyDown(upKey)) {
31      GetComponent<Rigidbody2D>().velocity=Vector2.up*speed;
32      spawnWall();
33    }
34    if(Input.GetKeyDown(downKey)) {
35      GetComponent<Rigidbody2D>().velocity=-Vector2.up*speed;
36      spawnWall();
37    }
38    if(Input.GetKeyDown(leftKey)) {
39      GetComponent<Rigidbody2D>().velocity=-Vector2.right*speed;
40      spawnWall();
41    }
42    if(Input.GetKeyDown(rightKey)) {
43      GetComponent<Rigidbody2D>().velocity=Vector2.right*speed;
44      spawnWall();
45    }
46     fitColliderBetween(wall,lastWallEnd,transform.position);
47    }
48
49  void spawnWall(){
50    lastWallEnd=transform.position;
51
52    GameObject c= Instantiate(wallPrefab,transform.position,Quaternion.
        identity);
53    wall=c.GetComponent<Collider2D>();
54  }
55
56  void fitColliderBetween(Collider2D what, Vector2 a, Vector2 b){
57    what.transform.position=a+(b-a)*0.5f;
58    float distance = Vector2.Distance(a,b);
59    if(a.x != b.x){
60      what.transform.localScale=new Vector2(distance+1,1);
```

```
61      }
62      else {
63        what.transform.localScale=new Vector2(1,distance+1);
64      }
65
66    }
67
68    void OnTriggerEnter2D(Collider2D co) {
69      if (co != wall) {
70          print("Player lost: " + name);
71          Destroy(gameObject);
72      }
73  }
74  }
```