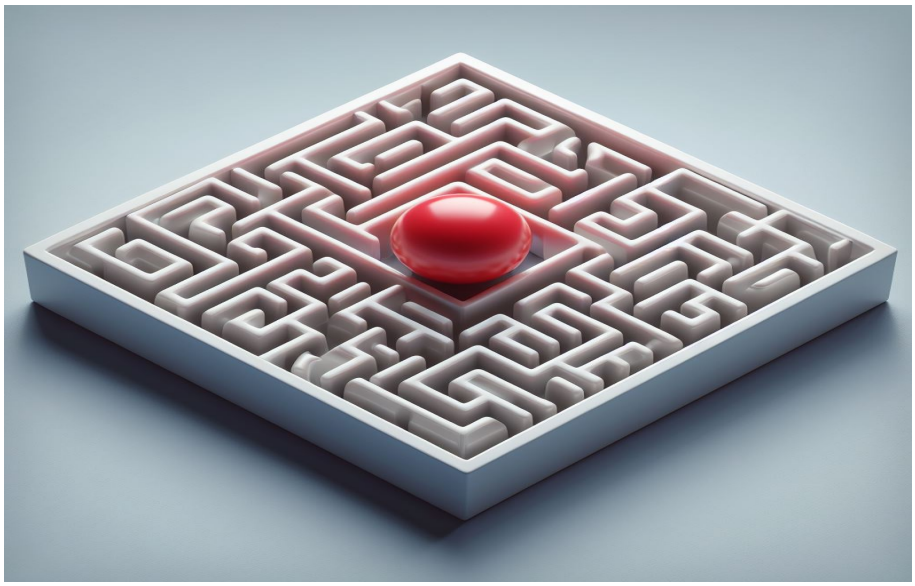
SŠPU
STŘEDNÍ ŠKOLA PRŮMYSLOVÁ
A UMĚLECKÁ, OPAVA

ZÁVĚREČNÁ STUDIJNÍ PRÁCE

dokumentace

MazeRoller



Autor: Aleš Najser

Obor: 18-20-M/01 INFORMAČNÍ TECHNOLOGIE
se zaměřením na počítačové sítě a programování

Třída: IT4

Školní rok: 2023/24

Poděkování

Poděkovat bych chtěl hned několika lidem. Jmenovitě:

- Marek Lučný - konzultace ohledně úvodní myšlenky a usměrnění cílů.
- Marcel Godovský - konzultace ohledně hardwarových částí projektu.
- David Kanovský - pomoc při studování C#.
- Jiří Ryba - pomoc s vytvořením obalu na hardwarové součástky.

Prohlášení

Prohlašuji, že jsem závěrečnou práci vypracoval samostatně a uvedl veškeré použité informační zdroje.
Souhlasím, aby tato studijní práce byla použita k výukovým a prezentačním účelům na Střední průmyslové a umělecké škole v Opavě, Praskova 399/8.

V Opavě 1. 1. 2024

. . . . . .

Podpis autora

Abstrakt

Tento projekt v Unity využívá algoritmus pro generování labyrintu, ve kterém hráč ovládá kuličku pomocí hardwarového ovladače. Hráč se snaží dostat kuličku do cíle labyrintu. Samotný labyrint má možnost dvou velikostí a to 10x10 a 20x20.

Se zvětšeným labyrintem se zvýší počet překážek a kamera se oddálí, aby bylo vidět celé hrací pole. Labyrint je taky opatřen generací cíle, který se může generovat pouze s určitou vzdáleností od hráče a určitého počtu překážek.

K projektu je také hardwarový ovladač složen z arduina a akcelerometru. Hra automaticky detekuje port na kterém je ovladač připojen a poté odesílá informace o naklonění ovladače přímo do unity, kde se s pomocí těchto informací pohybuje kulička v labyrintu.

Klíčová slova

Unity, 3D, Hardware, Hra, Arduino, Labyrint, Depth-first algoritmus

Obsah

Úvod

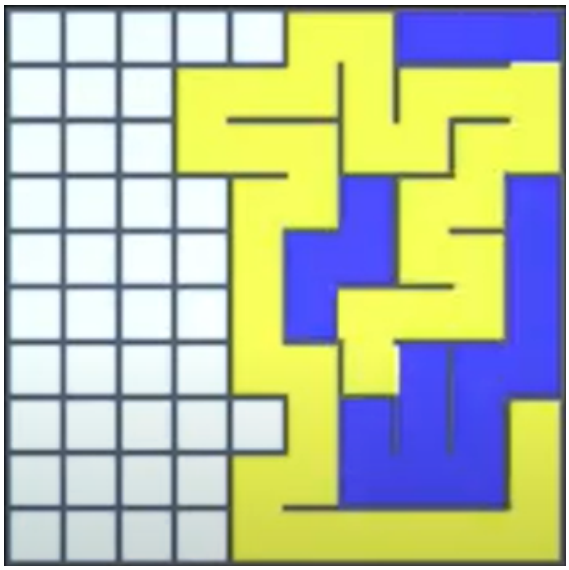
Unity je vynikající platforma pro vytváření 2D i 3D her, což získalo popularitu díky bezplatné licenci, kterou si může přisvojit kdokoli, dokud z jeho tvorby nevynese více než 100,000$ ročně. Tento limit byl nedávno zvýšen na 200,000$ za rok.
Jeho uživatelsky přívětivé prostředí umožňuje snadnou tvorbu her pomocí grafického rozhraní. V případě potřeby složitějších funkcí si však uživatel může vytvořit vlastní C# skripty, což umožňuje kompletní kontrolu nad vývojem.
Mě osobně fascinuje vývoj her a zejména jejich funkčnost. Toužil jsem pochopit, jak lze propojit vytvořený ovladač s počítačem a využívat ho k ovládání prvků her. Tímto rozhodnutím jsem se rozhodl tuto problematiku prozkoumat na vlastní kůži a vytvořit vlastní hru, která by poskytla unikátní herní zážitek díky originálnímu ovladači.
Dokumentace je rozdělena do čtyř kapitol:

- První kapitola obsahuje srozumitelné vysvětlení algoritmu použitého pro vytvoření náhodného labyrintu a jeho praktické využití.
- Druhá kapitola popisuje proces generace objektů v labyrintu.
- Třetí kapitola se věnuje použitým součástkám a vysvětluje, proč a jak byly určité součástky využity.
- Čtvrtá kapitola detailně popisuje propojení programu Unity se sériovým portem Arduina a jejich vzájemnou komunikaci.

1 GENEROVÁNÍ LABYRINTU

1.1 RECURSIVE BACKTRACKER ALGORITMUS

"Recursive backtracker"algoritmus.Labyrint je generován díky takzvanému "recursive backtracker"algoritmu. Algoritmus funguje následovně : představme si že máme velkou mřížku (v případě mého projektu 10x10 nebo 20x20). V této mřížce si vybereme náhodnou polohu jedné škatulky.Z této škatulky dále vybereme další kostičku do které budeme postupovat. Nyní si tento pohyb na druhou kostku poznamenáme ať víme jak jsme se tam dostali a odebereme stěnu mezi těmito kostkami. A znovu se opakuje tento proces dokud nenarazíme na tzv. "slepou uličku", odkud se nemáme kam pohnout. V tomto případě se pomalu vracíme po naší cestě zpět dokud nenajdeme další kostku do které můžeme postoupit a označíme již prošlou cestu jako hotovou, jelikož již nemá žádné možné cesty. Tento proces se opakuje dokud algoritmus neprojde všechny možné cesty, tudíž se bude muset vrátit zpět na startovní políčko a ukončí se.



Obrázek 1.1:
Vizualizace algoritmu, kde žlutá znamená políčko s jedním navštívením, modré políčko znázorňuje hotovou cestu (2 návštěvy) a bílé políčka ještě nebyly navštíveny.

1.1.1 Vytvoření mřížky labyrintu

Pro využití teorie již zmíněného algoritmu budeme muset sepsat třídu pro generování labyrintu. Nejdříve ale potřebujeme následovně vytvořit mřížku:

Kód 1.1: C# Vytvoření nodů (mřížky)

```
List<MazeNode> nodes = new List<MazeNode>();
        for (int x = 0; x < size.x; x++)
        {
            for (int y = 0; y < size.y; y++)
            {
                Vector3 nodePos = new Vector3(x - (size.x / 2f),
                    0, y - (size.y / 2f));
                MazeNode newNode = Instantiate(nodePrefab,
                    nodePos, Quaternion.identity, transform);
                nodes.Add(newNode);

            }
        }
```

1.1.2 Vyhledání možné cesty

Jakmile je vytvořena mřížka můžeme začít hledat cestu. K tomuto je potřeba zjistit kterým směrem se můžeme a naopak nemůžeme pohnout. Takto lze například zjistit zda políčko napravo od naší momentální pozice vyhovuje našim podmínkám pro další pohyb:

Kód 1.2: C# příklad hledání cesty

```
if(currentNodeX < size.x - 1)
{
    if (!completedNodes.Contains(nodes[currentNodeIndex + size.y
        ])&& !currentPath.Contains(nodes[currentNodeIndex + size.
        y]))
    {
        possibleDirections.Add(1);
        possibleNextNodes.Add(currentNodeIndex + size.y);
        }
}
```

1.1.3 Odstranění stěny při pohybu skrz labyrint

Při pohybu skrz labyrint je třeba odstranit stěnu pro vygenerování cesty. Toto je řešeno pomocí scriptu NodeState, který příjme status nodu (škatulky) a dále provede předdefinovanou akci. ,

Kód 1.3: C# Nastavení NodeState

```
1     switch (state){
2         case NodeState.Finish:
3             floor.GetComponent<MeshRenderer>().material.color =
                  Color.blue;
4             floorWithHole.gameObject.SetActive(false);
5             this.gameObject.tag = "Finish";
6             break;
7         case NodeState.Obstacle:
8             floor.gameObject.SetActive(false);
9             break;
10        }
11    }
```

2 Generace objektů v labyrintu

2.1 Náhodné vytvoření cíle v labyrintu

Každý labyrint by měl mít svůj konec v tomto projektu je náhodné generování labyrintu řešeno podobným způsobem jako generování překážek o kterém se znovu zmíním v další sekci. kód pro generaci cíle vypadá následovně:

Kód 2.1: C# Vytváření náhodných překážek v labyrintu

```
1  MazeNode randomCompletedNodeFinish = completedNodes[Random.Range
       (0, completedNodes.Count)];
2
3          // Calculate the distance between the object's position
               and Vector3.ZERO
4          float distanceToZero = Vector3.Distance(
               randomCompletedNodeFinish.transform.position, Vector3
               .zero);
5
6          // Choose a random completed node to place the object as
               long as it's at least 3 units away from Vector3.ZERO
7          while(distanceToZero < 3){
8              randomCompletedNodeFinish = completedNodes[Random.
                   Range(0, completedNodes.Count)];
9
10             distanceToZero = Vector3.Distance(
                   randomCompletedNodeFinish.transform.position,
                   Vector3.zero);
11         }
12
13         // Change the color of the chosen node for better
               visibility of the finish node
14         randomCompletedNodeFinish.SetState(NodeState.Finish);
```

2.2 Náhodné vytvoření překážek v labyrintu

Také je potřeba v labyrintu generovat náhodně překážky, aby nebyl příliš jednoduchý. Kód musí být opatřen aby se nevytvořila překážka na startu ani konci labyrintu.

Kód 2.2: C# Vytváření náhodných překážek v labyrintu

```
for(int i = 0; i<(size.x * size.y / 10); i++){
    MazeNode randomCompletedNode = completedNodes[Random.Range(0,
        completedNodes.Count)];
    if(randomCompletedNode.transform.position !=
       randomCompletedNodeFinish.transform.position &&
       randomCompletedNode.transform.position != Vector3.zero){
        randomCompletedNode.SetState(NodeState.Obstacle);
    }
}
```

2.3 Death bariéra

Jakmile máme v labyrintu překážky díky které může hráč vypadnout mimo labyrint, je potřeba přidat možnost restartovat hru od začátku. Přesně proto jsou potřeba death bariéry, které hráči oznámí že hru prohrál a pokud chce pokračovat musí začít od znovu. Toto jsem vyřešil pomocí C# skriptu který detekuje zda se hráč dotýka objektu který má tag "DeathBorder":

Kód 2.3: C# skript pro detekování hráče v death borderu

```
if (gameObject.CompareTag(deathBorderTag)){
            if (other.CompareTag(playerTag))
            {
                SceneManager.LoadScene("DeathMenu");
            }
        }
```

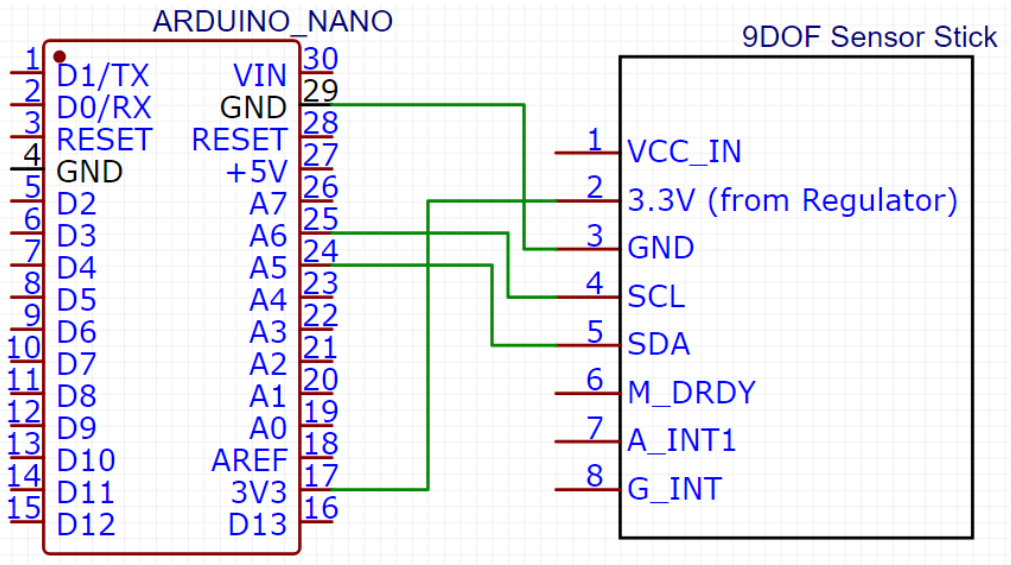
3 HARDWARE

3.1 VYUŽITÉ SOUČÁSTKY

Pro účely tohoto projektu jsou zapotřebí pouze 2 součástky:

3.1.1 Arduino nano

Arduino je využito pro zpracování a posílání dat akcelerometru. Tento konkrétní model arduina jsem si vybral pro jeho skvělý poměr cena/výkon. Zde je náhled na zapojení desky GY-85 která obsahuje akcelerometr:



Obrázek 3.1:
Schéma zapojení senzoru GY - 85

3.1.2 9 DOF senzor - GY-85

Tento model senzoru jsem si původně vybral pro využití gyroskopu, ale v průběhu projektu bylo jasné že nefunguje perfektně. Gyroskop na této desce totiž vypisuje stupňe za sekundu. Zpracování této hodnoty je složité a nepřesné, tudíž není gyroskop ideální pro

tento typ ovladače. Místo toho jsem proto použil akcelerometr který je přítomen na stejné desce. Díky knihovny pro GY-85 bylo programování velice snadné. Naklonění ovladače se tak dalo jednoduše vypočítat přes vzorec za pomoci proměnných přečtených z akcelerometru:

Kód 3.1: C++ - Arduino kód pro vypočítání naklonění

```
1 rolldeg = 180 * (atan(Y / sqrt(X * X + Z * Z))) / PI;
```

4 Spojení hardware a unity

4.1 Automatické detekování portu arduina

Abychom v unity mohli příjmout data z arduina musíme nejprve otevřít serial port ,přes který poté budeme číst. C# má díky knihovně System možnost přečíst všechny připojené "COM porty"pomocí "SerialPort.GetPortNames();".
Problém však je že to tyto available porty hodí do listu. Za předpokladu že uživatel používá pouze jeden ovladač jsem použil for loop který postupně zkouší všechny available porty dokud nenajde nějaký ze kterého lze číst. (arduino)

Kód 4.1: C# skript pro automatické nalezení komunikačního portu arduina

```
1  // list all serial ports
2  ports = SerialPort.GetPortNames();
3      for (int i = 0; i < ports.Length; i++){
4              portName = ports[i];
5              data_stream = new SerialPort(portName, 9600);
6              Debug.Log(portName);
7              // open port that got found
8              try
9              {
10                 data_stream.Open();
11                 portFound = true;
12                 break;
13             }
14             catch (Exception e)
15             {
16                 Debug.Log("Failed to open port " + portName + ":
                       " + e.Message);
17             }
18         }
19
20     if (!portFound)
21     {
22         Debug.LogError("Failed to find a valid COM port.");
23     }
24 }
```

4.1.1 Čtení dat z portu arduina a nastavení rychlosti hráče

Následující kód převezme data z arduina a nastaví je jako force v unity, čímž pohne tělesem, v našem případě objektem hráče.

Kód 4.2: C# skript pro přečtení a nastavení rychlosti pro hráče

```
1     //read arduino data
2         receivedString = data_stream.ReadLine();
3         string[] datas = receivedString.Split(','); //split the
              data between ','
4         rb.AddForce(0, 0, float.Parse(datas[1]) * sensitivity *
              Time.deltaTime, ForceMode.VelocityChange);
5         rb.AddForce(float.Parse(datas[0]) * sensitivity * Time.
              deltaTime, 0, 0, ForceMode.VelocityChange);
```

ZÁVĚR

Cílem projektu bylo vytvořit automaticky generovaný labyrint s automatickou generací cíle. Tento cíl jsem splnil, dokonce jsem během projektu přidal časti hry které mě z počátku nenapadly, jako například hardwarový ovladač a automatické generování překážek, které jsem také splnil. Práce mi také umožnila naučit se programovat v jazyce C# a uplatnit tyto znalosti v praxi. Dále jsem se naučil pracovat s knihovnami pro arduino.
V budoucnu bych rád přidal:

- Pohyblivé překážky - momentální překážky jsou příliš statické a proto jsou lehce překonatelné pro hráče. Proto bych rád přidal překážky které by následovali náhodně generovanou cestu, tím by se obtížnost hry určitě zvýšila.
- Více obtížností - v aktuální verzi hry jsou pouze dvě obtížnosti, konkrétně 10x10 a 20x20. V budoucnu bych chtěl přidát další obtížnosti, které by si hráč musel postupně odemnkout pokořením předešlých obtížností.
- Body v labyrintu - Hráč má momentálně jediný cíl a tím je se dostat do cíle. S přídáním bodů, nebo klíčů které by odemkly cíl potom co by je hráč posbíral by se stala hra trochu zábavnější a těžší.

SEZNAM POUŽITÝCH ZDROJŮ

[1] Depth-first search wikipedia. [online]. Aktualizováno 23.11. 2023. [cit. 2024-01-11]. Dostupné z: https://en.wikipedia.org/wiki/Depth-first_search.

[2] C# reference. [online]. © Microsoft 2023. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/. [cit. 2024-01-11].

[3] Arduino forum. [Online]. © 2020 Arduino. Dostupné z: https://forum.arduino.cc/. [cit. 2024-01-11].

[4] Arduino stack exchange. [online]. © 2024 Stack Exchange Inc. Dostupné z: https://arduino.stackexchange.com/. [cit. 2024-01-11].

[5] Copilot tvůrce obrázků. [online]. © 2024 Microsoft. Dostupné z: https://copilot.microsoft.com/images/create. [cit. 2024-01-11].

Seznam obrázků