Тимски проект по Визуелно програмирање
Maze Game

1. Опис на апликацијата

Апликацијата која ја работивме е игра во лавиринт која што се вика Maze Game и се состои од неколку нивоа.

Освен едноставниот дизајн што ја поседува играта, имплементиравме и неколку алгоритми со цел да предизвика желба да се игра повеќе, некои алгоритми кои имаат одреден број до колку пати можеш да ја играш соодветната нивоа и така натаму.

2. Упатство за користење

2.1 Нова игра

Во почетниот прозорец(слика1) на играта има можност да се започнува играта(New Game) а потоа се појавува следниот прозор(слика2) каде може да се одлучи која нивоа сакате да си ја играте базирано на тежина:

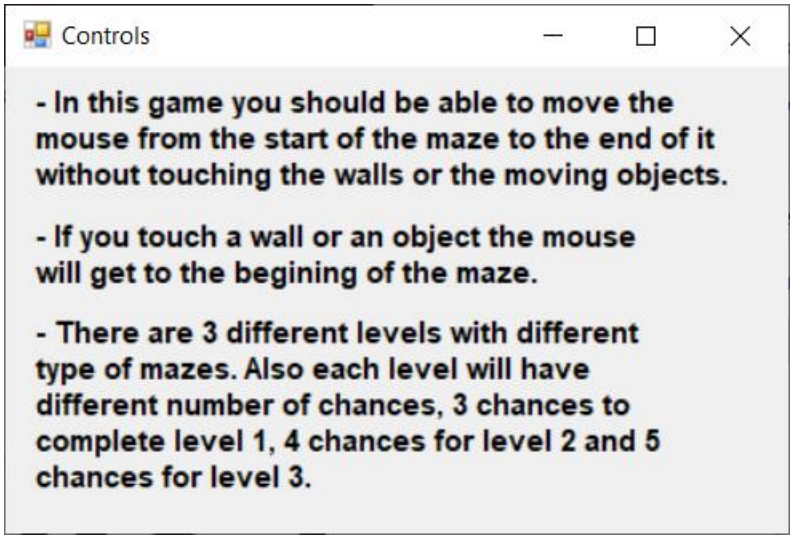
- Easy
- Normal
- Hard

Слика1

Слика2

2.2 Прозорец за објаснување на начинот на играње и правилата за играта

Во почетниот прозорец(слика1) по притискање на копчето за контроли(Controls) се отвара нов прозорец(слика3) во кој е опишано како се игра играта и некои можности за победување на нивоата.



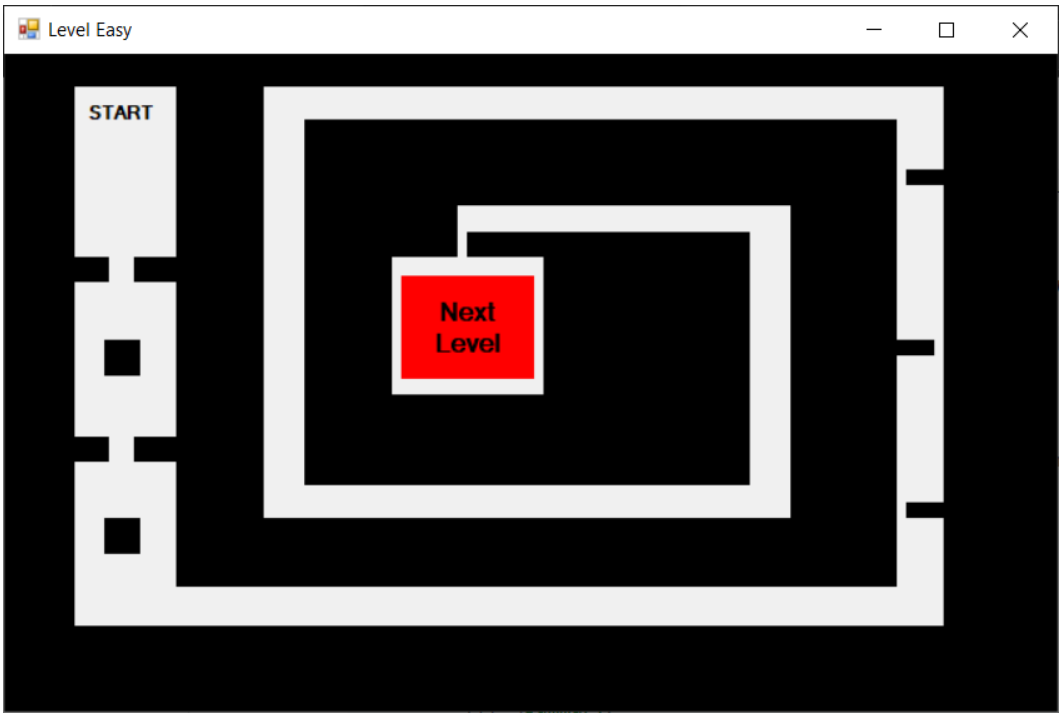
Слика3

Правила

Во ова игра играчот треба да е способен да го премести глувчето од почетокот на лавиринтот до крајот без да ги допирате ѕидовите или објектите кои што се во движење.

Ако играчот допре некои од ѕидовите или објект со глувчето, глувчето се враќа во почетокот на лавиринтот.

Како што кажавме претходно играта се состои од три нивоа базирано на тежина и секоја ниво има различен број на обиди, лесната ниво има 3 обиди, нормалната ниво има 4 обиди и тешкото ниво има 5 обиди за играње се доколку не победувате.



Слика4

2.3 Кредити

Во почетниот прозорец(слика1) по притискање на копчето за кредити(Credits) се отвара нов прозорец(слика5) каде се појавуваат нашите имиња.

Слика5

3. Претставување на проблемот

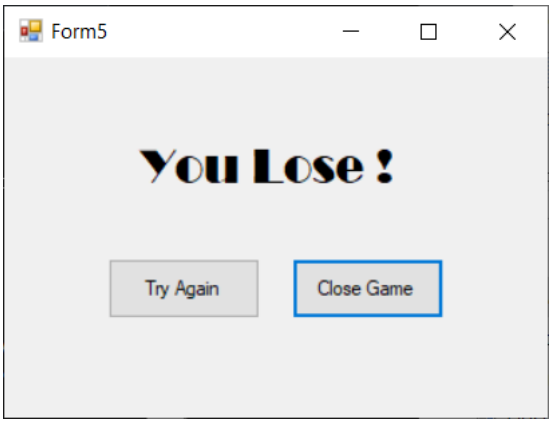
3.1 Податочни структури

Главните функции со кои почнува програмата се чуваат во класата Program и од неа почнуваат да се извршуваат и другите функции на другите класи кои сите наследуваат од класата Form и се приближо слични, на пример класите Form3, Form4 и Form6 ги имат некои од овие функции:

- `private void wall(object sender, EventArgs e)`
  Оваа функција ги пресметува бројот на дозволени обиди за нивоата и доколку не е надминат дозволениот број се повикува moveToStart(), во спротивно се појавува прозорец(слика6) за пак играње или затворање на играта.

```
private void tryAgain(object sender, EventArgs e)

private void closeGame(object sender, EventArgs e)
```



Слика6

- `private void moveToStart()`
  Ако ги допрете ѕидовите ова функција глувчето го праќа на почетокот на лавиринтот.

- `private void nextLevel(object sender, EventArgs e)`
  Оваа функција прикажува текст после победување на нивото и те испраќа на следното ниво.

- `private void checkPoint(object sender, EventArgs e)`
  Оваа функција ги пресметува бројот на дозволени обиди за Normal или Hard нивоата доколку играчот стигна до CheckPoint и ако не е надминат дозволениот број на обиди се повикува moveToChackPoint(), во спротивно се појавува прозорец(слика6) за пак играње или затворање на играта.

- `private void moveToChackPoint()`
  Ако ги допрете ѕидовите ова функција глувчето го праќа кај CheckPoint на лавиринтот.

3.2 Класата Hard

MoveObstacle

Креиравме една класа „ObstacleConfiguration“ во Hard формата со конструктор кој прифаќа:

- објектот кој што треба да се движи
- Насоката дефинирана со enum типот

```
public enum MoveDirection
    {
        Left,
        Right,
        Top,
        Bottom
    }
```

- DistanceMoved – секогаш 0
- DistanceEnd – до каде треба да оди објектот
- Step – колку брзо ќе оди објектот

Hard форма кога ќе се иницијализира, креиравме еден moveTimer кој што на секои 50 милисекунди ќе ја извршува настанот „MoveTimeEvent“.

За секоја конфигурација во ObstacleConfiguration се повикува MoveObstacle, во овој метод објектот продолжува да се движи сè додека DistanceMove<DistanceEnd, кога ќе стигне до DistanceEnd насоката се менува и DistanceMoved се рестартира на 0.

```
private void MoveObstacle(ObstacleConfiguration obstacleConfiguration)
        {
            if (obstacleConfiguration.DistanceMoved < obstacleConfiguration.DistanceEnd)
            {
                this.Invoke((MethodInvoker)delegate ()
                {
                    var step = obstacleConfiguration.Step;
                    switch (obstacleConfiguration.MoveDirection)
                    {
                        case MoveDirection.Left:
                            obstacleConfiguration.Control.Left -= step;
                            break;
                        case MoveDirection.Right:
                            obstacleConfiguration.Control.Left += step;
                            break;
                        case MoveDirection.Top:
                            obstacleConfiguration.Control.Top -= step;
                            break;
                        case MoveDirection.Bottom:
                            obstacleConfiguration.Control.Top += step;
                            break;
                    }

                    obstacleConfiguration.DistanceMoved += step;

                });
            }
            else
            {
                switch (obstacleConfiguration.MoveDirection)
                {
                    case MoveDirection.Left:
                        obstacleConfiguration.MoveDirection = MoveDirection.Right;
                        break;
                    case MoveDirection.Right:
                        obstacleConfiguration.MoveDirection = MoveDirection.Left;
                        break;
                    case MoveDirection.Top:
                        obstacleConfiguration.MoveDirection = MoveDirection.Bottom;
                        break;
                    case MoveDirection.Bottom:
                        obstacleConfiguration.MoveDirection = MoveDirection.Top;
                        break;
                }
                obstacleConfiguration.DistanceMoved = 0;
            }
        }
```