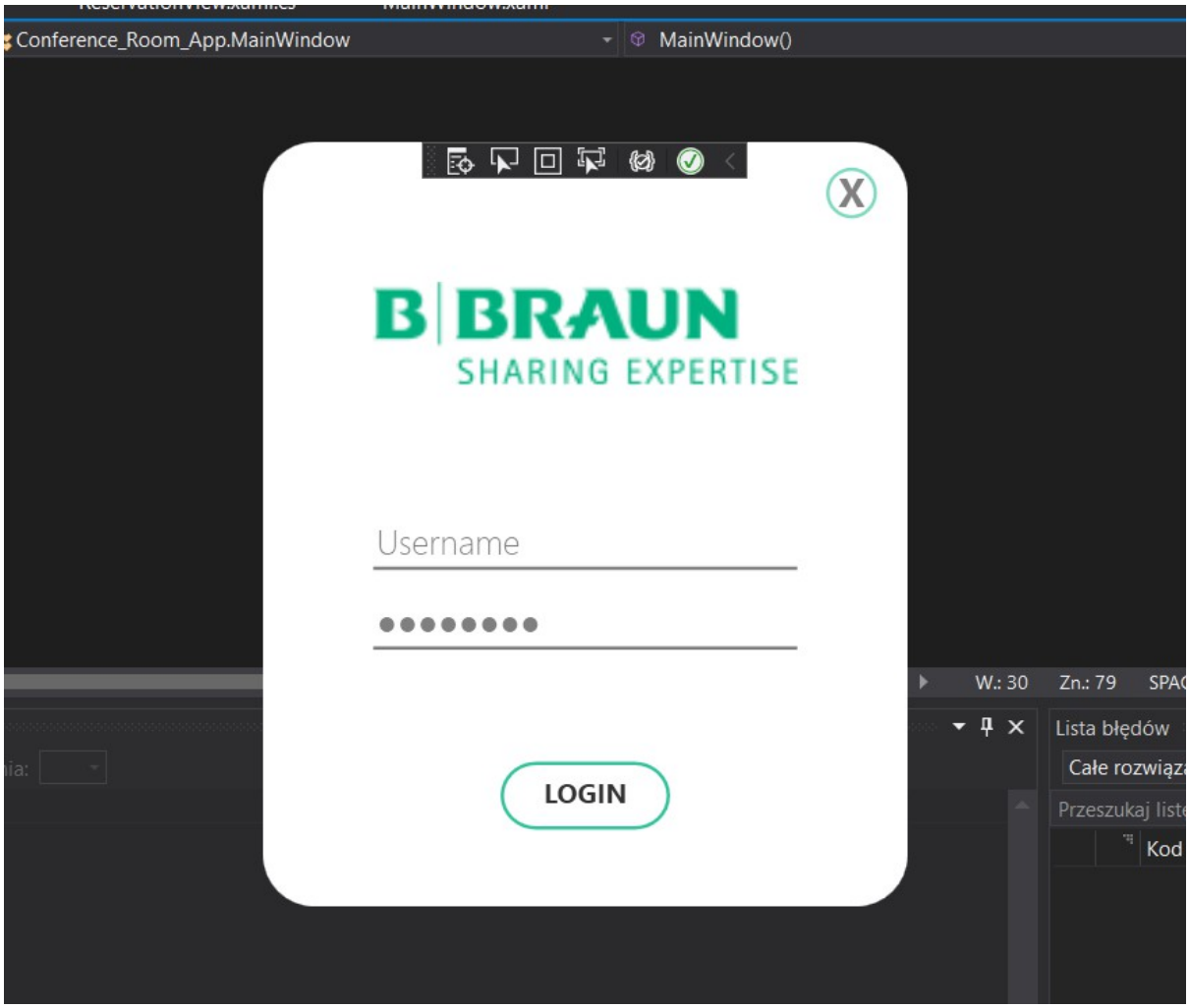
Paweł Wawrzyniak

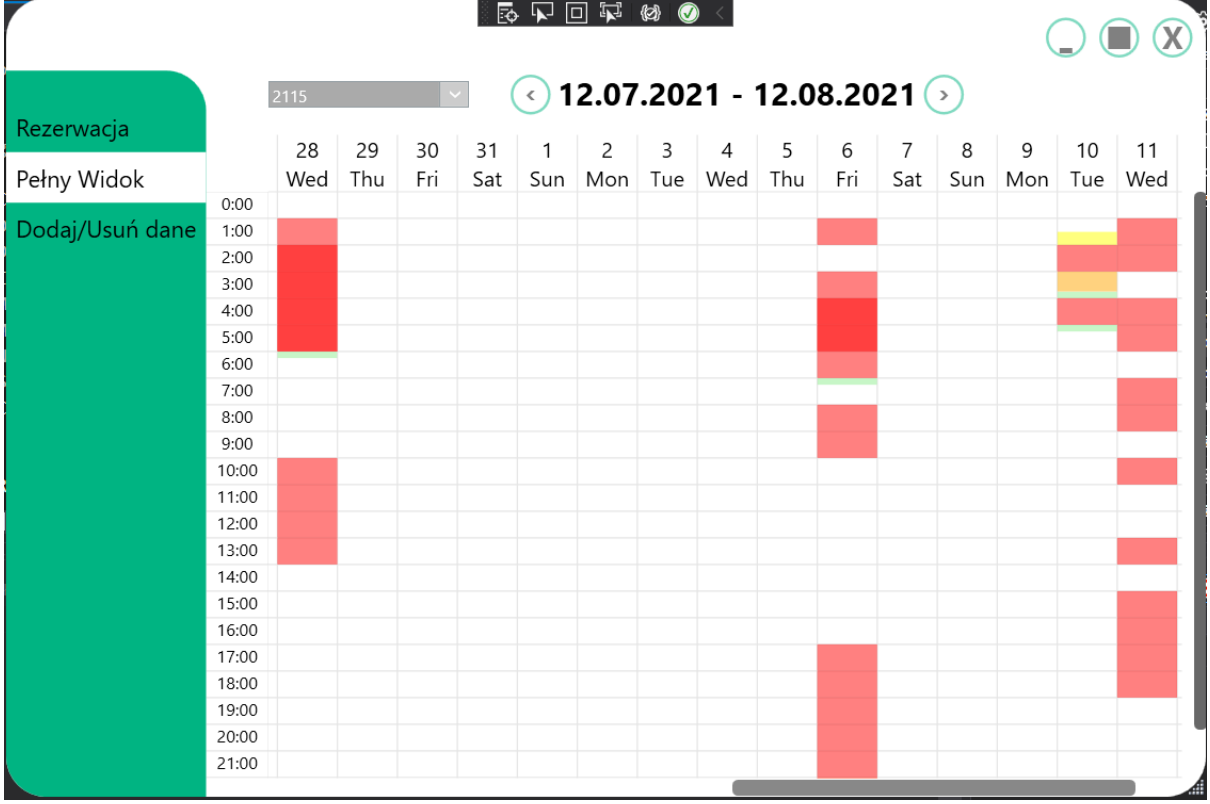
Portfolio - Programista

Zaprojektowanie i wykonanie pełnego programu

Rezerwacja Sal



Grid kalendarza :




```
public void createGrid()
{
    int daysQuantity = DateTime.DaysInMonth(choosenDate.Year, choosenDate.Month) + 1;
    int cellWidth = 45;
    int cellHeight = 20;
    int hours = 24;
    setDataRange();

    for (int i = 0; i < hours; i++)
    {
        RowDefinition hourRows = new RowDefinition()
        {
            Height = new GridLength(cellHeight)                         //Tworzenie wierszy z taką samą wysokością w dwóch różnych tabelach (główny grid i geid godzin)
        };
        RowDefinition rows = new RowDefinition()
        {
            Height = new GridLength(cellHeight)
        };

        myGrid.RowDefinitions.Add(rows);                                //Dodawanie wierszy do (parent) odpowiednich tabel
        hourGrid.RowDefinitions.Add(hourRows);
    }

    for (int i = 0; i < daysQuantity; i++)
    {
        ColumnDefinition dataCols = new ColumnDefinition()
        {
            Width = new GridLength(cellWidth),                          //Tworzenie kolumn z taką samą szerokością w dwóch różnych tabelach(główny grid i grid daty)
        };

        ColumnDefinition cols = new ColumnDefinition()
        {
            Width = new GridLength(cellWidth)
        };
        myGrid.ColumnDefinitions.Add(cols);                             //Dodawanie kolumn do (parent) odpowiednich tabel
        dataGrid.ColumnDefinitions.Add(dataCols);
    }

    for (int i = 0; i < hours; i++)
    {
        TextBlock hourLbl = new TextBlock()
        {
            HorizontalAlignment = HorizontalAlignment.Center,
            VerticalAlignment = VerticalAlignment.Bottom,
            TextAlignment = TextAlignment.Center,
            FontSize = 13,
            Height = cellHeight,                                        //Dodawanie trextblock`ów do tabeli ze spisem godzin (grid)
            Width = cellWidth,
            Foreground = Brushes.Black,
            Text = i.ToString() + ":00"
        };
        Grid.SetRow(hourLbl, i);
        hourGrid.Children.Add(hourLbl);
    }

    for (int i = 0; i < daysQuantity; i++)
    {
        TextBlock dataLbl = new TextBlock()
        {
            FontSize = 16,
            Foreground = Brushes.Black,
            Text = choosenDate.AddDays(i).Day.ToString() + "\n" + choosenDate.AddDays(i).DayOfWeek.ToString().Remove(3),
            TextAlignment = TextAlignment.Center,
            TextWrapping = TextWrapping.Wrap                            //Dodawanie labeli do tabeli ze spisem dni, pobieranie i modyfikacja tekstu do wymagań (dni)
        };
        Grid.SetColumn(dataLbl, i);
        dataGrid.Children.Add(dataLbl);
    }
```

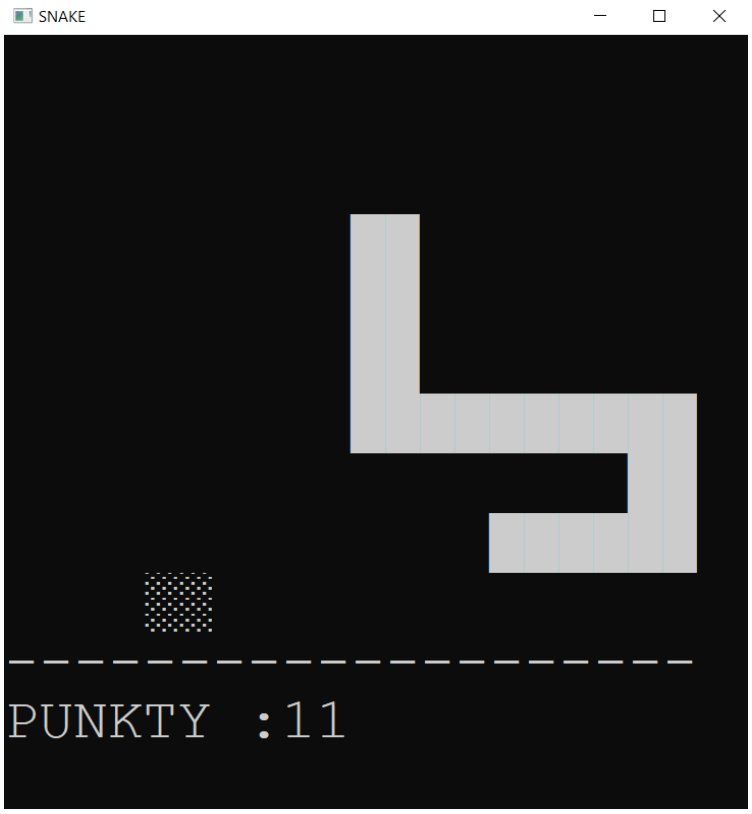
Najtrudniejszym zadaniem w tej aplikacji było wyświetlenie własnego gridu kalendarza.

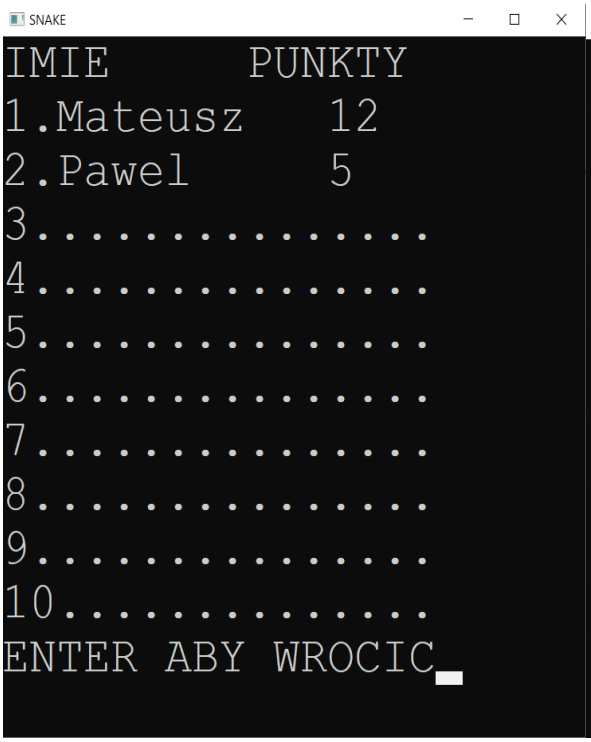
Zaprojektowanie tej aplikacji wymagało ode mnie użycia całej mojej wiedzy na temat .Net oraz baz danych.

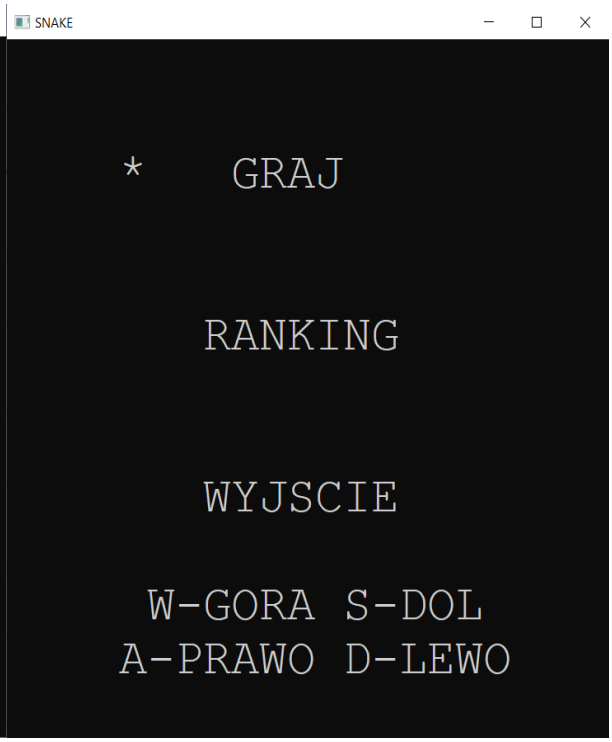
Snake 2020r.

Został wykonany w terminalu bez silnika - 272 linijki

github: https://github.com/mrskaterr/Snake







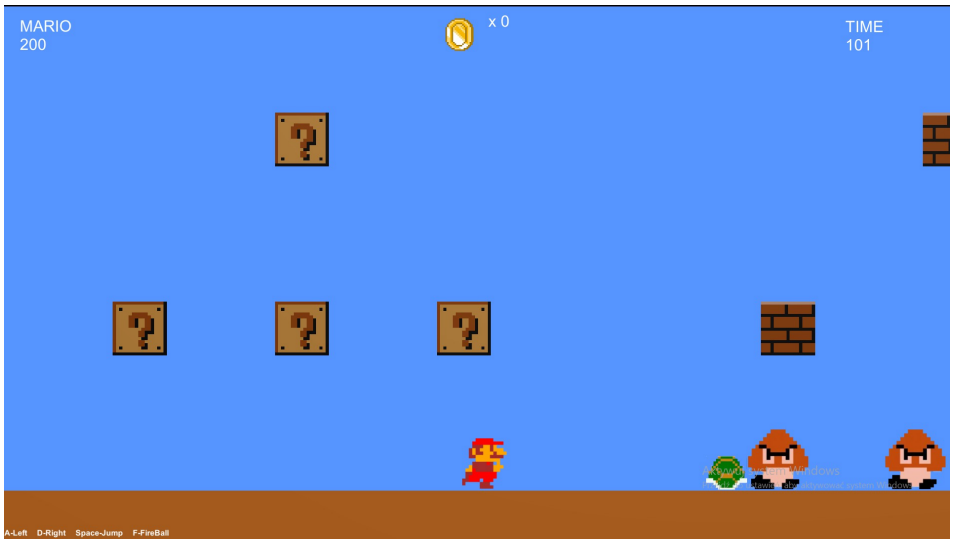
```
142 Snake* turnAndBuild(char turn, Snake* start) {
143     Snake* Help = new Snake;
144     if (turn == 'w' || turn == 'W') {
145         if (HeadX == 0)HeadX = Size - 1;
146         else HeadX--;
147     }
148     else if (turn == 's' || turn == 'S') {
149         if (HeadX == Size - 1)HeadX = 0;
150         else HeadX++;
151     }
152     else if (turn == 'd' || turn == 'D') {
153         if (HeadY == Size - 1)HeadY = 0;
154         else HeadY++;
155     }
156     else if (turn == 'a' || turn == 'A') {
157         if (HeadY == 0)HeadY = Size - 1;
158         else HeadY--;
159     }
160
161     if (*(*(Space + HeadX) + HeadY) == Point) {
162         Score++;
163         *(*(Space + HeadX) + HeadY) = Body;
164         Help->BodyPart = (*(Space + HeadX) + HeadY);
165         Help->Next = start;
166         return Help;
167     }
168     else if (Score == 0) {
169         Help->BodyPart = (*(Space + HeadX) + HeadY);
170         *(*(Space + HeadX) + HeadY) = Body;
171         Help->Next = NULL;
172         (*(*start).BodyPart) = Blank;
173         start = NULL;
174         return Help;
175     }
176     else if (*(*(Space + HeadX) + HeadY) == Blank) {
177         Snake* help2 = new Snake;
178         Help->BodyPart = (*(Space + HeadX) + HeadY);
179         *(*(Space + HeadX) + HeadY) = Body;
180         Help->Next = start;
181         help2 = start;
182         for (int i = 0; i < Score; i++) {
183             start = help2;
184             if (help2 == NULL) {
185                 SaveScore();
186                 Ranking();
187             }
188             help2 = help2->Next;
189             if (help2->Next == NULL) {
190                 (*(*help2).BodyPart) = Blank;
191                 help2 = NULL;
192                 start->Next = NULL;
193                 return Help;
194             }
195         }
196     }
197     return NULL;
```

Nad tą częścią kodu przesiedziałem najwięcej czasu. Jest on odpowiedzialny za poruszanie się węża, wydłużanie się go oraz zdobywanie punktów.

MARIO 2020r.

Unity C#

github: https://github.com/mrskaterr/Mario



Pierwszy poziom Mario wykonany wyłącznie przeze mnie.

```
public class Turtle : MonoBehaviour
{
    public float Distance;
    public float Speed;
    float PrivateSpeed = 0;
    public GameObject Player;
    public GameObject Mushrom;
    public GameObject Canvas;

    Rigidbody2D rb;
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    void Update()
    {
        if (PrivateSpeed == 0 && (gameObject.transform.position.x - Player.transform.position.x) <= Distance) PrivateSpeed = Speed; //Run Aktivation
        rb.velocity = new Vector3(PrivateSpeed, 0, 0);
    }
    private void OnCollisionEnter2D(Collision2D collision)
    {

        if (collision.gameObject == Player && collision.gameObject.GetComponent<Rigidbody2D>().velocity.y < 0)//Tutrle transorm to Turtle2
        {
            GameObject.FindGameObjectWithTag("Canvas").GetComponent<TimeScoreCoin>().AddScore(200);
            gameObject.transform.GetChild(0).gameObject.SetActive(false);
            gameObject.transform.GetChild(1).gameObject.SetActive(true);
            if (collision.gameObject.transform.position.x > gameObject.transform.position.x) gameObject.GetComponent<Turtle>().PrivateSpeed =Mathf.Abs(PrivateSpeed)*1.5f ;//RIGHT
            if (collision.gameObject.transform.position.x < gameObject.transform.position.x) gameObject.GetComponent<Turtle>().PrivateSpeed =-Mathf.Abs(PrivateSpeed)*1.5f;//LEFT


        }
        else if (collision.gameObject == Player && collision.gameObject.transform.localScale.x > 1)
        {
            collision.gameObject.GetComponent<Movement>().Jump2();
            collision.gameObject.transform.localScale = new Vector3(1, 1, 1);
        }
        else if (collision.gameObject == Player)//Player DIE
        {
            collision.gameObject.GetComponent<PlayerDead>().enabled = true;
        }
        else if (collision.gameObject == Mushrom && PrivateSpeed != Speed)//Mushrom Die
        {
            collision.gameObject.SetActive(false);
            Canvas.GetComponent<TimeScoreCoin>().AddScore(200);
        }
        else if (rb.velocity.x == 0) PrivateSpeed *= -1;  //Change
    }
}
```

Najtrudniejszym zadaniem było stworzenie movementu dla żółwia. Żółw mógł Cię zaatakować lecz gdy dobrze się na niego skoczy, unieszkodliwia się go i można posługiwać się nim jak piłką do gry oraz atakować inne potwory.