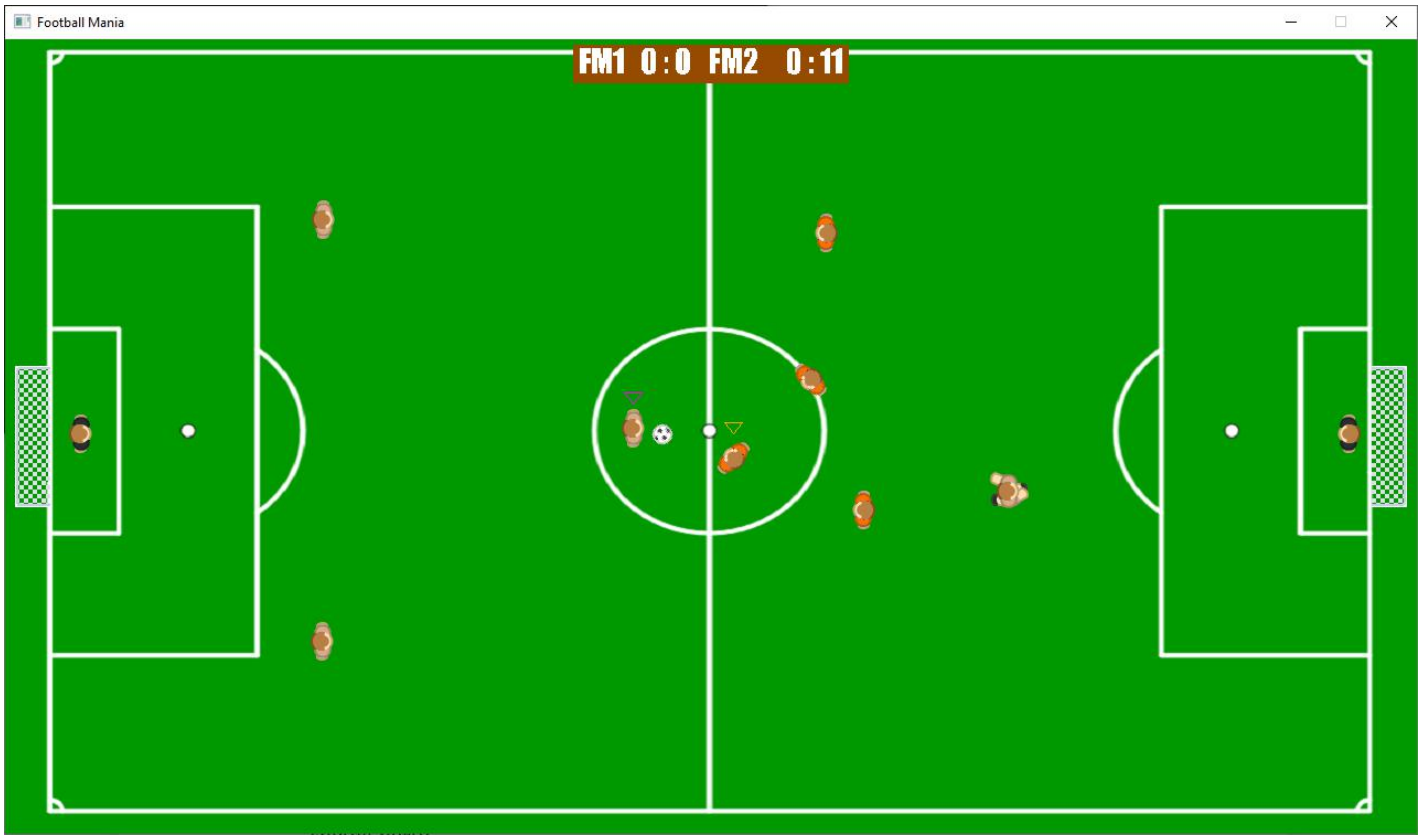
# Football Mania


## Budacă Marius-Andrei

Faculty of Automatic Control and Computer Engineering, Iasi

email: <marius-andrei.budaca@student.tuiasi.ro>,

academic year: <2021-2022>


- **Gameplay:** *Jocul este un te tipul offline multiplayer, astfel incat este nevoie de 2 persoane pentru a fi jucat. Gamplay-ul consta in demonstrarea abilitatilor intr-un simulator primitiv de fotbal. Obiectivul jocului este de a inscrie mai multe goluri in porta inamica decat adversarul vostru.*

- **Plot:** *2 echipe rivale dintr-un mic oras se intalnesc fata in fata. Du-ti echipa pe culmile gloriei si invinge rivalul in acest meci de campionat!*
-
- **Characters:** ... (*ex:*
  - ***Portarii*** *au rolul de a impiedica mingea sa intre in poarta echipei*
  - ***Aparatorii*** *au rolul de a impiedica atacantii inamici de a avea posesie sau un culoar de sut spre spatiul portii*
  - ***Atacantii*** *au rolul de a inscrie in poarta inamica, fiind si cei mai vitezomani din echipa*

- **Mechanics**
    - W/A/S/D sau UP/DOWN/LEFT/RIGHT pentru a misca jucatorii selectati. Restul jucatorilor se vor misca automat
    - K sau NUM2 pentru a pasa mingea celui mai apropiat coechipier aflat in directia in care jucatorul selectat inainteaza
    - L sau NUM3 pentru a suta mingea spre o poarte. Atentie! Daca veti suta spre directia propriei porti este posibil sa inscrieti un auto-gol.
    - E sau NUM_ENTER pentru a schimba jucatorul selectat atunci cand va aparati

**La evaluare se vor avea in vedere urmatoarele:**

| # | Criteriu | Realizat |
|---|---|---|
| 1 | Abstractizare | 10 |
| 2 | Încapsulare | 10 |
| 3 | Moștenire (ierarhie de grad 3 minim) | 10 |
| 4 | Polimorfism | 9 |
| 5 | Interfețe (clase abstracte) | 10 |
| 6 | Gestionarea erorilor (excepții) | 9 |
| 7 | Salvarea sau încărcarea configurației jocului (Lucrul cu fișiere) | 10 |
| 8 | Număr de niveluri cu dificultate graduală (minim 3) | 8 |

Cod pe Git: https://github.com/AndreiBudaca3/Proiect-POO

## 1. Abstractizare

Campurile claselor sunt membri privati, ascunsi in afara namespace-ului clase:

```
class Player : public Object
{
private:
    int currentFrame;
    int frameAdd;
    int frameNumber;
    int frameTime;
    int team;
    bool movingState;
    int verticalMovement;
    int horizontalMovement;
    double dx, dy;
    bool selected;
    int switchDelay;
    bool noUpdate;
    int shootCharge;

    SDL_Rect spriteCoord[3];
    Texture* pTex;

public:
```

```
class Scoreboard : public Object
{
private:
    int team1Score;
    int team2Score;
    int frameNumber;
    scoreTime* time;
    std::string team1Name;
    std::string team2Name;
    Texture* sTex;

public:
```

2. **Incapsulare**: campurile sunt modificate doar de metodele din clasa respectiva:

```
Texture* getTexture(void);
bool isMoving(void);
bool isSelected(void);
void setSelection(bool state);
void setMovingState(bool state);
void setSwitchDelay(int value);
int getSwitchDelay(void);
int getTeam(void);
double getDX(void);
double getDY(void);
void setDX(double dx);
void setDY(double dy);
bool willNotUpdate(void);
void setUpdate(bool state);
void resetAnimation(void);
void setShootCharge(int shootCharge);
```

```
class scoreTime
{
private:
    int minutes;
    int seconds;
public:
    scoreTime();
    scoreTime(int seconds);
    ~scoreTime();

    void addSeconds(int seconds = 1);
    void subSeconds(int seconds = 1);
    bool isZero(void);

    int getMinutes(void);
    int getSeconds(void);
};
```

3. **Mostenire**

Un exemplu de ierarhie de 3 clase: clasa Goalkeeper mosteneste clasa Player, care la randul ei mosteneste clasa Obect

```
class Goalkeeper : public Player
{
private:
```

```
class Player : public Object
{
private:
```

4. **Polimorfism** - destructorii claselor mostenite sunt declarate ca functii virtuale

```
virtual ~Player();
```

```
virtual ~Object();
```

5. **Interfete** - avem 2 clase pur virtuale:

Clasa Object:

```
    virtual void draw(SDL_Point* center = NULL, SDL_RendererFlip flip = SDL_FLIP_NONE) = 0;
    virtual void update(SDL_Event* e = NULL, const Uint8* keyboardState = 0, Object** obj = NULL, bool* flag = NULL) = 0;
};
```

Clasa Player:

```
void draw(SDL_Point* center = NULL, SDL_RendererFlip flip = SDL_FLIP_NONE);
virtual void update(SDL_Event* e = NULL, const Uint8* keyboardState = 0, Object** obj = NULL, bool* flag = NULL) = 0;
```

6. **Gestionarea erorilor**: in cazul in care apare o eroare la incarcarea unei texturi, programul nu mai poate rula. Eroarea va fi afisata pe ecran pentru cateva secunde inainte ca programul sa se inchida singur.

```
TTF_Font* gFont = TTF_OpenFont("testAssets/impact.ttf", 50);
if (gFont == NULL)
{
    throw(std::string("Failed to load font!"));
    return;
}
//Render text surface
SDL_Surface* textSurface = TTF_RenderText_Solid(gFont, text.c_str(), SDL_Color{0xFF, 0xFF, 0xFF, 0xFF});
if (textSurface == NULL)
{
    throw(std::string("Unable to render text surface!"));
    return;
}
else
{
    //Create texture from surface pixels
    tex = SDL_CreateTextureFromSurface(r, textSurface);
    if (tex == NULL)
    {
        throw(std::string("Unable to create texture from rendered text!"));
        return;
    }
```

```
try
{
    backgorund = new Texture(std::string("testAssets/terrain.png"), wRenderer);

    objVec[PLAY_BUTTON] = new Button(wRenderer, std::string("testAssets/start.png"), new SDL_Point{ 586, 180 }, 108, 48, MENU_FLAG, false);
    objVec[RESUME_BUTTON] = new Button(wRenderer, std::string("testAssets/resume.png"), new SDL_Point{ 586, 275 }, 108, 48, PAUSE_FLAG, false);
    objVec[CUSTOMISE_BUTTON] = new Button(wRenderer, std::string("testAssets/hist.png"), new SDL_Point{ 586, 370 }, 108, 48, RIGHT_TEXTURE_FLAG);
    objVec[QUIT_BUTTON] = new Button(wRenderer, std::string("testAssets/quit.png"), new SDL_Point{ 586, 465 }, 108, 48, QUIT_FLAG);
    objVec[CONTROLS_BUTTON] = new Button(wRenderer, std::string("testAssets/controls.png"), new SDL_Point{ 586, 275 }, 108, 48, LEFT_TEXTURE_FLAG);
    objVec[MENU_BUTTON] = new Button(wRenderer, std::string("testAssets/menu.png"), new SDL_Point{ 586, 370 }, 108, 40, MENU_FLAG);
}
catch (std::string err)
{
    clearScreen();
    std::cout << err << '\n';

    SDL_Rect b{ 0, 0, 1280, 720 };
    SDL_SetRenderDrawColor(wRenderer, 0x01, 0x01, 0x01, 0xFF);
    SDL_RenderFillRect(wRenderer, &b);

    Texture error('t', err, wRenderer);
    error.renderTexture();
    updateVisuals();
    SDL_Delay(1000);

    quitGame = true;
}
```

7. **Salvarea sau încărcarea configuraţiei jocului** - din fisierul config.ini sunt incarcate date despre caracteristicile vizuale ale playerilor, cat si despre lungimea meciului. De altfel in aceleasi fisier vor fi salvate istoricul meciurilor, istoric din care vor fi preluate ultimele 10 meciuri.

```
    std::ifstream fin("config.in");

    char buffer[30];
    fin >> buffer >> player1color;
    fin >> buffer >> player2color;
    fin >> buffer >> player1Name;
    fin >> buffer >> player2Name;
    fin >> buffer >> matchLength;
    player1Diffculty = 0;
    player2Diffculty = 0;

    nrHist = 0;
    bool overflow = false;

    while (fin.getline(buffer, 256))
    {
        hist[nrHist++] = std::string(buffer);
        if (nrHist == 10)
        {
            nrHist = 0;
            overflow = true;
        }
    }

    if (overflow) nrHist = 10;

    fin.close();
}
```

```
    std::ofstream fout;
    fout.open("config.in", std::ios_base::app);

    fout << team1Name << ' ' << team1Score << '-' << team2Score << ' ' << team2Name << '\n';

    if (team1Score > team2Score) flag[TEAM1_WIN_FLAG] = true;
    if (team1Score < team2Score) flag[TEAM2_WIN_FLAG] = true;

    fout.close();
}
```

8. **Număr de niveluri cu dificultate graduală -** pe masura ce playerii vor castiga meciuri, jucatorii lor se vor misca mai greu. Astfel va creste dificultatea de a castiga un meci impotriva adversarului. Dificultatea se reseteaza in momentul in care jocul se inchide.

```
            if (flag[TEAM1_WIN_FLAG])
            {
                game->increasePlayer1Difficulty();
                flag[TEAM1_WIN_FLAG] = false;
            }
            if (flag[TEAM2_WIN_FLAG])
            {
                game->increasePlayer2Difficulty();
                flag[TEAM2_WIN_FLAG] = false;
            }
        }
```

```
objVec[TEAM1_PLAYER2] = new Defender(wRenderer, player1color, new SDL_Point{ 265, 145 }, 1, player1Diffculty);
objVec[TEAM1_PLAYER3] = new Defender(wRenderer, player1color, new SDL_Point{ 265, 530 }, 1, player1Diffculty);
objVec[TEAM1_PLAYER4] = new Striker(wRenderer, player1color, new SDL_Point{ 620, 285 }, 1, player1Diffculty, 90, true);
objVec[TEAM1_PLAYER5] = new Striker(wRenderer, player1color, new SDL_Point{ 620, 380 }, 1, player1Diffculty, -90);
```