Projektna dokumentacija

IGRA SAH

Predmet: CS233 – Programiranje 3D igara

Školska godina 2021/2022

Ime i prezime studenta: Aleksandar Golubović

Broj indeksa: 4550

Profesor: Miljan Milošević

Asistent: Vuk Mihailovcić Takimoto

1. UVOD

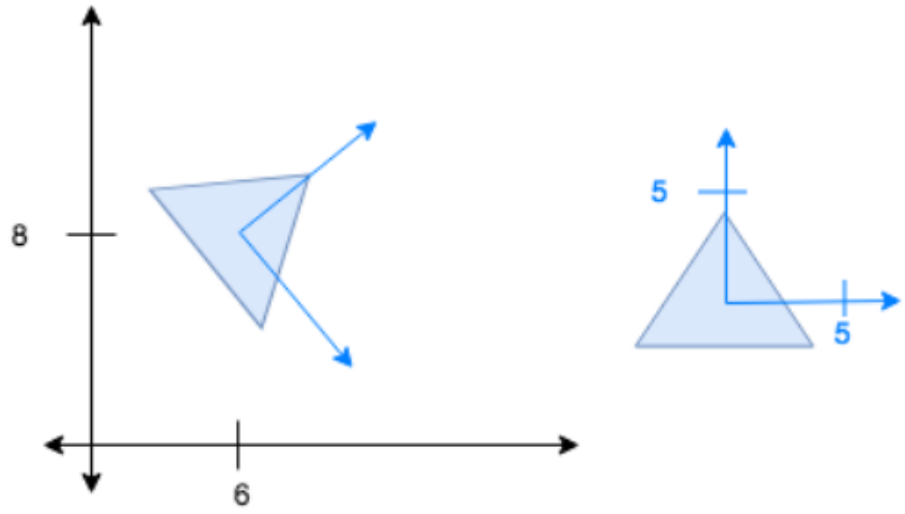
Postavka zadatka:

Za predmet CS233 Programiranje 3D igara, projektni zadatak podrazumeva jednostavnu video igru u trodimenzionalnom prostoru. Igra se razvija u programskom jeziku C/C++ i za grafički prikaz mora da koristi aplikacioni programski interfejs OpenGL i biblioteku GLUT. Igra mora biti sinhronizovana u realnom vremenu. Potrebno je implementirati pristupe razvoja igre koji se uče u okviru predmeta da bi se napravila kompletna igra.

OpenGL i GLUT:

Grafički sistem korišćen u ovom projektu je OpenGL. Iako se često zove biblioteka, OpenGL je zapravo aplikacioni programski interfejs koji omogućava komunikaciju sa grafičkim procesorom i čija forma je standardizovana među proizvođačima grafičkih kartica. Razlika nije toliko ni bitna, i OpenGL može da se smatra bibliotekom i praktičnom smislu, jer pruža rutine pomoću kojih 4 Slika 3. Petlja igre koristeći delta vreme programer može da diktira šta će biti prikazano na ekranu. Jedino bude očigledno da zaista nije biblioteka kada se dovede do ponašanja koja nisu definisana standardom i rezultati budu drugačiji u zavisnosti od implementacije proizvođača.

OpenGL ima određene sposobnosti koje su pogotovo pogodne za izradu 2D igre. Koordinati sistem koji definiše OpenGL daje infrastrukturu za smeštanje elemenata u okviru igre. Programeri imaju mogućnost da manipulišu taj koordinatni sistem i da diktiraju kako će biti prikazan u prozoru, odnosno koji deo. Nad koordinatnim sistemom je moguće raditi linearne transformacije što drastično olakšava iscrtavanje jer je moguće, uz manje računanja, pomerati i okretati elemente. Transformacije su takođe veoma korisne za prevođenje akcija iz globalnog koordinatnog sistema u lokalni.



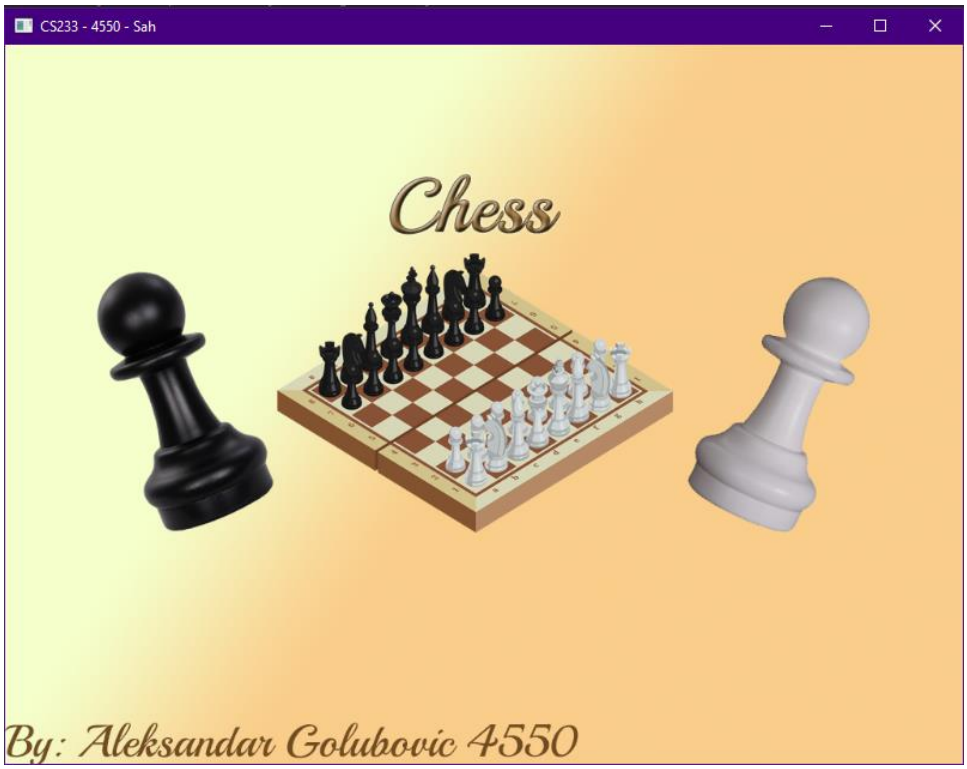
Slika 1.1 Globalni i lokalni koordinatni sistem

Kao što može da se vidi na slici 4, sa leve strane je prikazan globalni koordinatni sistem sa trouglom smeštenim na tački (6, 8). Trougao je okrenut pod uglom od 30° od x-ose. Da bi se ovaj trougao iscrtao u globalnom koordinatnom sistemu, potrebno je izračunati njegova tri verteksa na osnovu njegovog centra i ugla po kojem je okrenut. Mnogo lakši pristup iscrtavanju je prvo vršenje translacije do tačke (6, 8) koja do te tačke pomeri koordinatni početak i zatim rotacija od 30° koja rotira ose tako da se preklapaju sa orijentacijom trougla.

2. OPIS ZADATKA

Kao rešenje projektnog zadataka izradjena je igra sah u 3D prostoru. Sah je strateska drustvena igra za dva igraca koja se igra na kariranoj tabli dimenzija 8x8. Tok igre se izvrsava tako sto igraci pomeraju jedan od svojih 16 komada (8 pesaka, 2 skakaca, 2 lovca, 2 topa, jedan kralj i jedna kraljica). Igraci se dele na dva tima, beo i crn, beli tim uvek ima prvi pokret. Igra se zarvsava tako sto igrac stavlja drugog u stanje sahmat, stanje gde je gubitnikov kralj pod napadom i bez slobodnih mesta. Ostala pravila se mogu procitati na [Pravila igre sah - Wikipedija.]

Premisa igre je jednostavna, nakon pocetnog “spash art” prozora igracu je prikazan pocetni meni gde moze da izabere izlazak, pregled kontrola igre kao i pokretanje same igre saha. U menijima korisnik bira zeljenju opciju pomocu vertikalnih strelica a svoj izbor potvrdjuje SPACE tasterom.

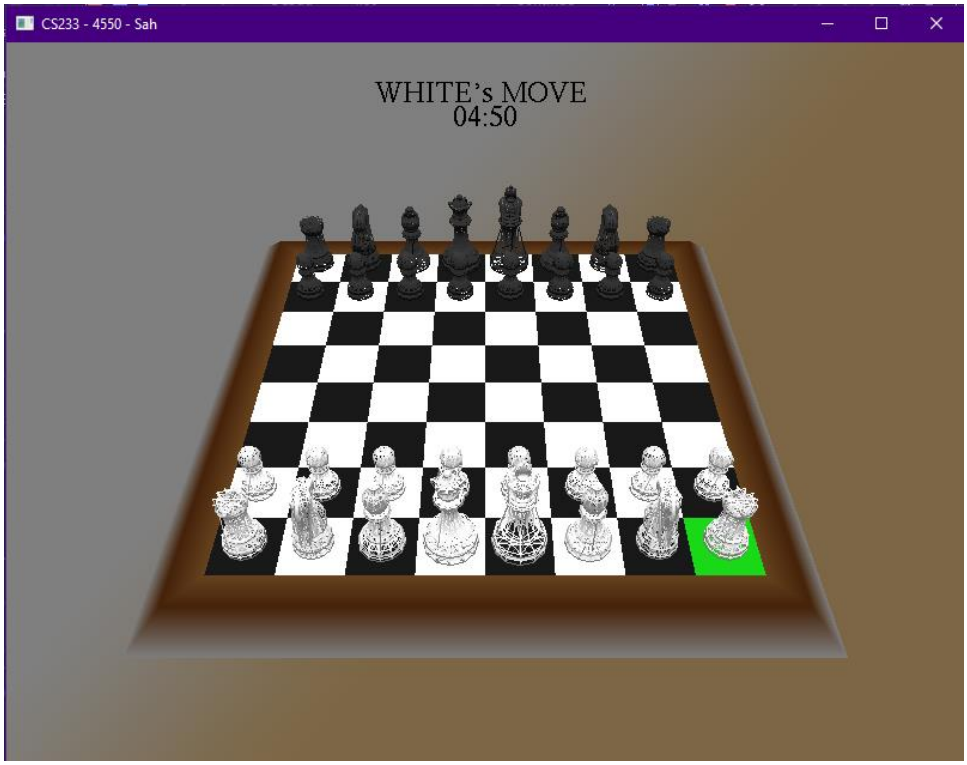


Slika 2.1 Pocetni prozor igre

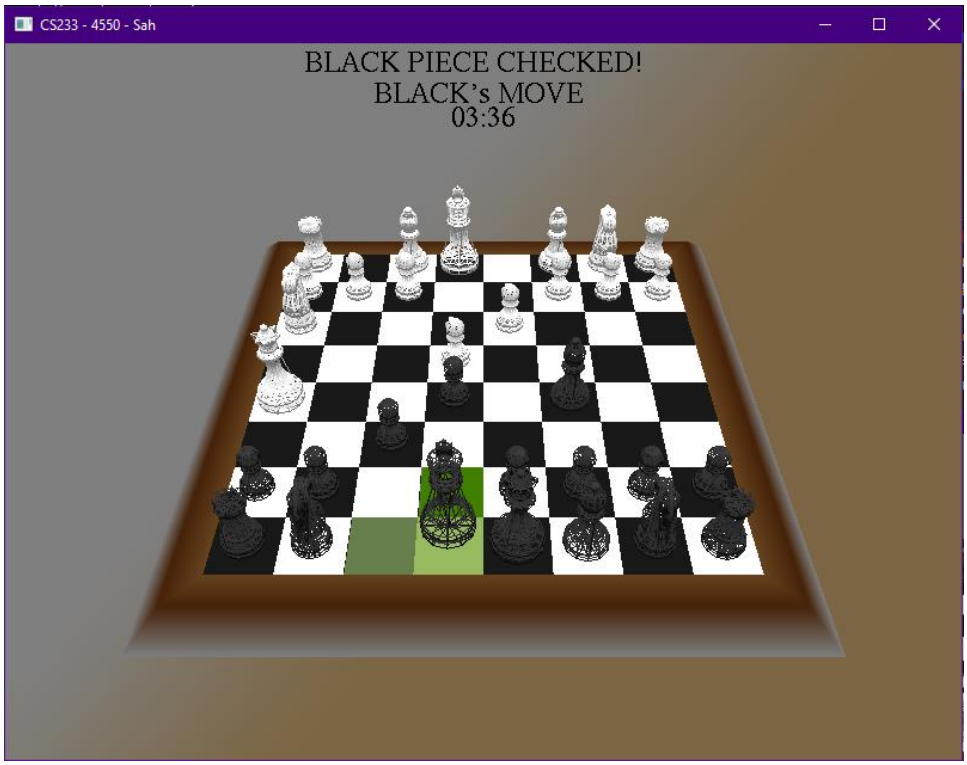
Slika 2.2 Glavni meni igre

Slika 2.3 Prikaz kontrola igre

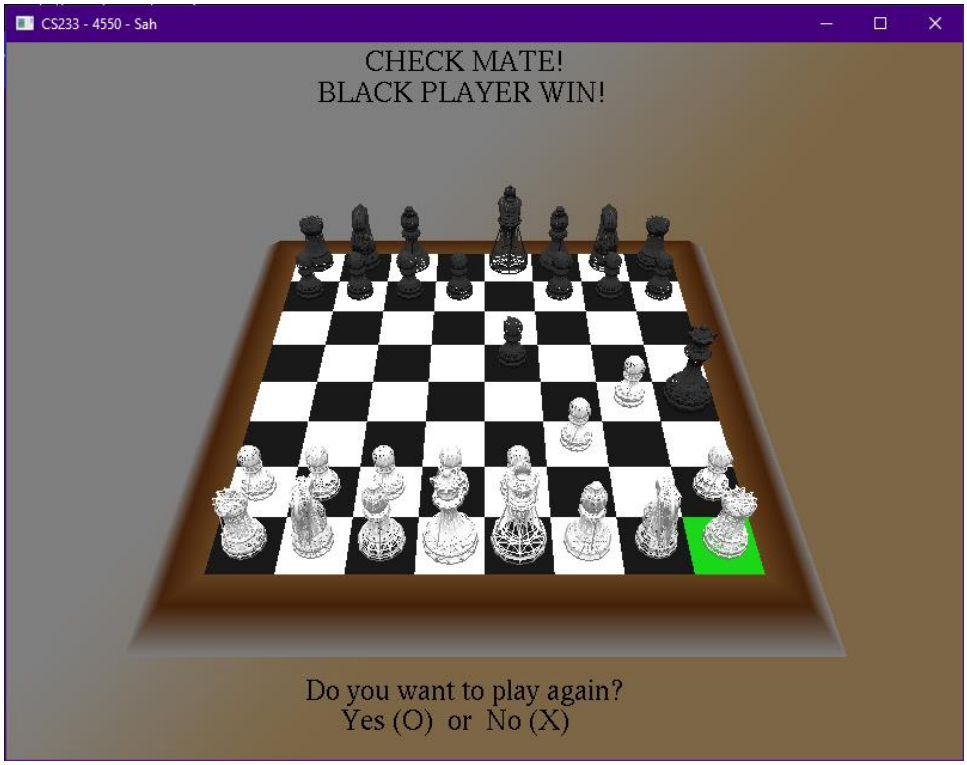
Igrac pomera figure dugmicima tastature (WASD) tako sto pomera kontrolni kursor i sa tasterom SPACE bira zeljenu figuru nakon cega mu se prikazuju svi moguci potezi tj. sva slobodna polje na koja izabrana figura moze da skoci. Igrac ima mogucnost pomeranja kamere(zumiranje i kretanje levo desno). Pored manuelnog pomeranja kamere od strane igraca, nakon svakog pokreta kamera se automatksi zarotira tako da bude na strani figura ciji je sledeci potez. Igrac ima mogucnost da restartuje i pauzira igru. Korisnicki interfejs tokom same igre sadrzi preostalo vreme kao i indikator igraca ciji je trenutni potez.



Slika 2.4 Izgled same igre sah



Slika 2.5 Crne figure su u stanju sah



Slika 2.6 Sah mat! Crni igrac pobedjuje

3. TEHNICKI OPIS

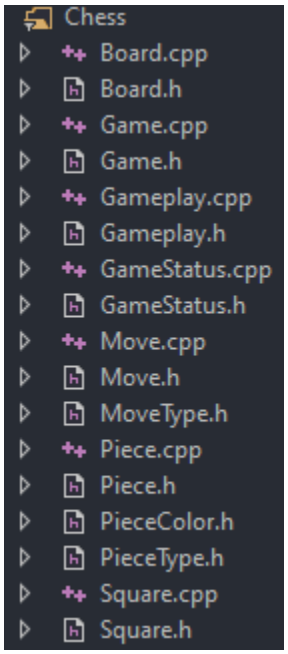
Radi snalazljivosti ovaj segment sam podelio u 3 dela:

- Logika igre sah

- Logika cele igre(modeli, slike, input...)

- Iscrtavanje svega(display funkcija)

-Logika igre sah

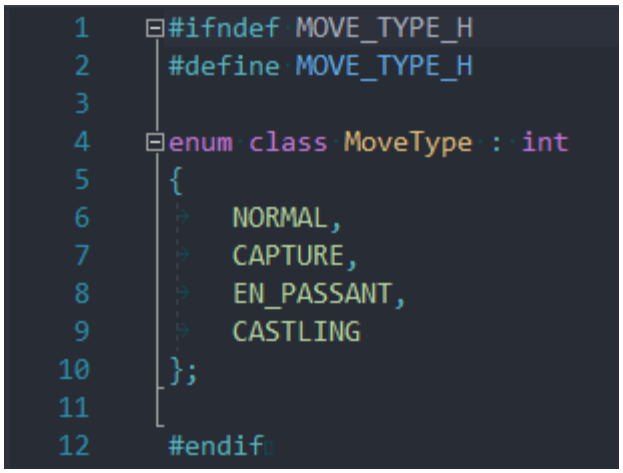


3.1.1 Lista klasa koriscena za logiku saha

Stanje igre cuvamo u klasi **Board** koja u sebi sadrzi dvodimenzionalni niz **Square**-ova dimenzija 10x10.

Klasa **Square** sadrzi figuru koja zauzima to polje, tj klasu **Piece.** Ona sadrzi sve informacije o figuri poput njene boje i tipa figure.

Klase **PieceColor**, **PieceType** i **MoveType** su enum klase koje korsitimo za definisanje **boja**, **tipova** i **tipova pokreta.**



```
#ifndef MOVE_TYPE_H
#define MOVE_TYPE_H

enum class MoveType : int
{
    NORMAL,
    CAPTURE,
    EN_PASSANT,
    CASTLING
};

#endif
```

3.1.2 Primer enum klase (MoveType)

Glavne klase za kontrolu igre su **Game** i **Gameplay.** Klasu **Gameplay** za definisanje mogucih kretanja svih figura, samo kretanje figura kao i vecinu logike za tok igre. Jedina klasa koju iz ove liste koristimo u glavnom programu je klasa **Game,** ona sadrzi instance klasa Gameplay, Board i GameStatus, log svih napravljenih pokreta kao i sve funkcije potrebne za tok igre sah(od kojih je vecina nasledjena iz klase **Gameplay)**. Klasa **GameStatus** sadrzi sve posebne pokrete poput en passant-a, rokada...

```
#include <stack>

#include "Piece.h"
#include "Gameplay.h"
#include "Board.h"
#include "GameStatus.h"
#include "Move.h"

class Game
{
    private:
        std::stack<Move> log;
        Gameplay* gameplay;
        Board* board;
        GameStatus* status;
        int turn;
    public:
        Game();
        void setInitialPieces(PieceColor color);
        std::stack<Move> getAllLog();
        Board* getBoard();
        GameStatus* getGameStatus();
        std::vector<Move> getValidMoves(int fromRow, int fromCol);
        bool move(int fromRow, int fromCol, int toRow, int toCol);
        PieceColor getPieceColor(int fromRow, int fromCol);
        Piece* getPiece(int fromRow, int fromCol);
        bool isSquareOccupied(int fromRow, int fromCol);
        bool inCheckState();
        bool inCheckMateState();
        int getTurn();
        void nextTurn();
        PieceColor getTurnColor();
        bool promote(int row, int col, PieceType type);
};
```

3.1.3 Header fajl glavne klase za sah(Game)

-Logika cele igre

Za kontrolisanje toka igre koristimo enumerator **GameState** koji sadrzi: **SPLASH, MENU, CONTROLS, IN_PROGRESS, MODE_SELECTION, PAUSED**. Pravimo globalnu instancu ovog enuma i u zavisnosti od trenutnog stanja mi menjamo njegovu vrednost. Tokom petlje igre definisani su tipovi ponasanje u zavisnosti od treutnog **GameState**-a.

Za modele samih figura koristimo klasu Model koja prima obj. fajl i iscrtava prosledjen model. Ona radi na principu prolazenja kroz obj. fajl gde dodaje sve informacije o modelu u vector **position_normal_texture** koji kasnije koristi za iscrtavanje istog.

```
void Model::Draw()
{
    int index;
    glPushMatrix();
    glBegin(GL_LINE_STRIP);
    for(int i=0; i<sz; i++)
    {
        index = 8 * i;
        glNormal3f(position_texture_normal[index+5], position_texture_normal[index+6], position_texture_normal[index+7]);
        glTexCoord2f(position_texture_normal[index+3], position_texture_normal[index+4]);
        glVertex3f(position_texture_normal[index], position_texture_normal[index+1], position_texture_normal[index+2]);
    }
    glEnd();
    glPopMatrix();
}
```

3.2.1 Funkcija za iscrtavanje modela u klasi **Model**

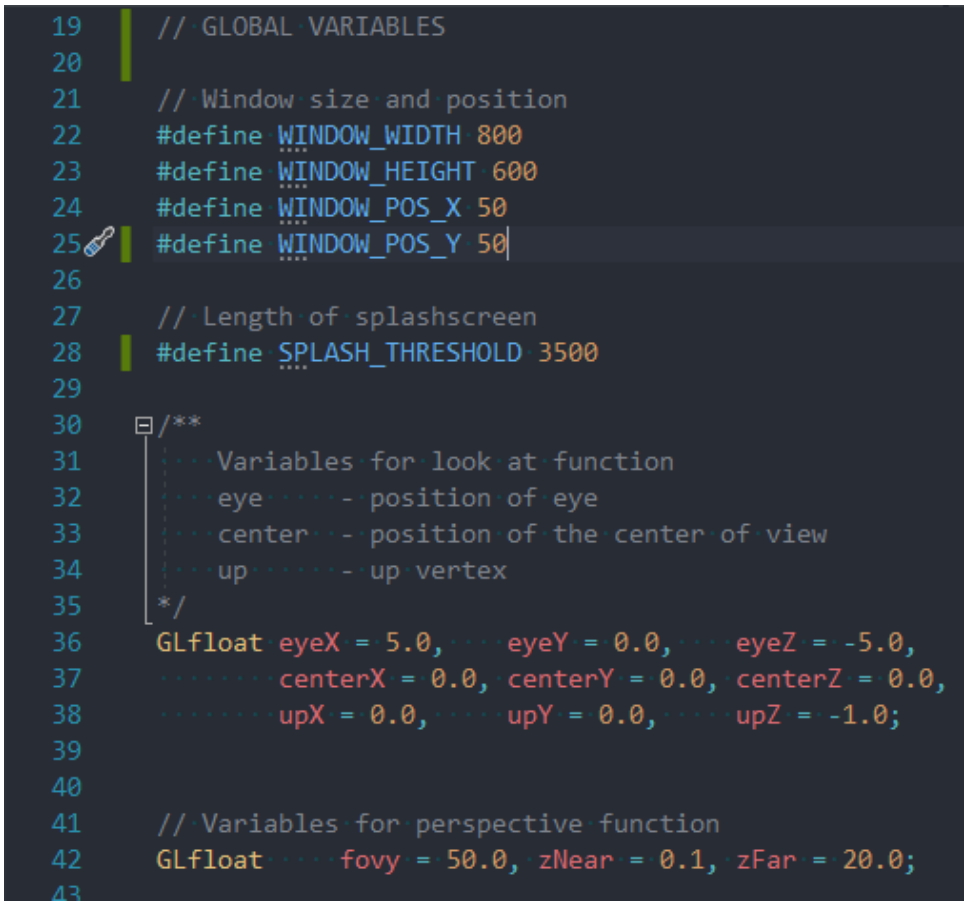
Za iscrtavanje korisnickog interfejsa koristimo klasu **Sprite** koja u sebi sadrzi sve moguce informacije i funkcionalnosti za rad sa slikama.

```
void initSprites() {
    splashScreen = new Sprite(1);
    splashScreen->SetFrameSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    splashScreen->SetNumberOfFrames(1);
    splashScreen->AddTexture("Sprites/splash.png", false);
    splashScreen->IsActive(true);
    splashScreen->IsVisible(true);
    splashScreen->SetPosition(0.0f, 0.0f);

    menuScreen = new Sprite(1);
    menuScreen->SetFrameSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    menuScreen->SetNumberOfFrames(1);
    menuScreen->AddTexture("Sprites/mainmenu.png", false);
    menuScreen->IsActive(true);
    menuScreen->IsVisible(true);
    menuScreen->SetPosition(0.0f, 0.0f);

    controlsScreen = new Sprite(1);
    controlsScreen->SetFrameSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    controlsScreen->SetNumberOfFrames(1);
    controlsScreen->AddTexture("Sprites/controls.png", false);
    controlsScreen->IsActive(true);
    controlsScreen->IsVisible(true);

    backgroundScreen = new Sprite(1);
    backgroundScreen->SetFrameSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    backgroundScreen->SetNumberOfFrames(1);
    backgroundScreen->AddTexture("Sprites/background.png", false);
    backgroundScreen->IsActive(true);
    backgroundScreen->IsVisible(true);
```

3.2.2 Inicijalizacija sprajtova klase **Sprite**

Funkciju keyboard() nam je glavna za kontrolisanje toka igre, ovde u zavisnosti od korisnikovog unosa i trenutne stanje igre radimo neophodne stvarni za glatak nastavak igre.

Na pocetku same main klase imamo definicije svih globalnih promenljivih koje su koriscene u tokom igre:



```
// GLOBAL VARIABLES

// Window size and position
#define WINDOW_WIDTH 800
#define WINDOW_HEIGHT 600
#define WINDOW_POS_X 50
#define WINDOW_POS_Y 50

// Length of splashscreen
#define SPLASH_THRESHOLD 3500

/**
    Variables for look at function
    eye     - position of eye
    center  - position of the center of view
    up      - up vertex
*/
GLfloat eyeX = 5.0,    eyeY = 0.0,    eyeZ = -5.0,
        centerX = 0.0, centerY = 0.0, centerZ = 0.0,
        upX = 0.0,     upY = 0.0,     upZ = -1.0;


// Variables for perspective function
GLfloat    fovy = 50.0, zNear = 0.1, zFar = 20.0;
```

3.2.3 Globalne promenljive 1

```
45    // Variables for light
46    GLfloat position[] =          { 0.0f, 0.0f, 100.0f, 0.0f };
47    GLfloat diffusion[] =         { 1.0f, 1.0f, 1.0f, 1.0f };
48    GLfloat normal_board[] =      { 0.0f, 0.0f, 1.0f };
49    GLfloat normal_valid_move[] = { 0.0f, 0.0f, -1.0f };
50    GLfloat mat_diffusion[] =     { 0.8, 0.8, 0.8, 1.0 };
51    GLfloat mat_specular[] =      { 0.1, 0.1, 0.1, 1.0 };
52    float   ang = 0;
53
54    // Variables for managing view
55    GLfloat screen_ratio, zoomOut = 2;
56
57
58    // Model Loading
59    Model Pawn   ("Models/Pawn.obj");
60    Model Rook   ("Models/Rook.obj");
61    Model Knight ("Models/Knight.obj");
62    Model Bishop ("Models/Bishop.obj");
63    Model King   ("Models/King.obj");
64    Model Queen  ("Models/Queen.obj");
65
66    // Variables for sprites
67    Sprite* splashScreen;
68    Sprite* menuScreen;
69    Sprite* controlsScreen;
70    Sprite* backgroundScreen;
71    Sprite* pausedScreen;
72    Sprite* modeScreen;
73    Sprite* arrowIcon;
```

3.2.4 Globalne promenljive 2

```
75    // Variables for sounds
76    FMOD::System* audioMgr;
77    FMOD::Sound* sfxClick;
78    FMOD::Sound* sfxTicking;
79    FMOD::Sound* sfxOver;
80    FMOD::Channel* backgroundChannel;
81    FMOD::Channel* sfxChannel;
82    bool isOverPlayed = false, isTickingPlayed = false; // used for preventing constant playing of sounds
83
84
85    // Array used for storing all the positions of the main menu arrow used for navigating
86    Sprite::Point arrowPositions[3] = { {445, 416}, {445, 353}, {445, 290} };
87    uint16_t arrowIndex = 2;
88
89    // Pre start
90    bool pressed = false;
91
92    // Game loading
93    Game* chess;
94    void newGame();
95
96
97    // Real-time variables
98    GameState gameState;
99    bool inGame = false, verify = false;
100   int selectedRow = 1, selectedCol = 1;
101   int moveToRow = 1, moveToCol = 1;
102   bool selected = false;
103   bool board_rotating = true;
104   int rotation = 0;
105   bool check = false, checkMate = false, timeOut = false;
106   bool closeGame = false;
107   bool needPromote = false;
```

3.2.5 Globalne promenljive 3

```
109   //                                                default time set to 5 mins          ---      unimplemented     ---
110   float elapsedTime = 0, startTime = 0, goalTime = 0, selectedTime = 300000, showTime = 0, whiteTime = 0, blackTime = 0;
111   float pausedTime = 0, unPausedTime = 0;
112   bool unlimitedGame = false;
113
114
115   // Chess board vertices
116   GLfloat     chessBoard[12][3] = {{-4.0, -4.0, 0.5},
117                                    {-4.0,  4.0, 0.5},
118                                    { 4.0,  4.0, 0.5},
119                                    { 4.0, -4.0, 0.5},
120
121                                    {-4.5, -4.5, 0.5},
122                                    {-4.5,  4.5, 0.5},
123                                    { 4.5,  4.5, 0.5},
124                                    { 4.5, -4.5, 0.5},
125
126                                    {-5.0, -5.0, 0.0},
127                                    {-5.0,  5.0, 0.0},
128                                    { 5.0,  5.0, 0.0},
129                                    { 5.0, -5.0, 0.0}};
130
```

3.2.6 Globalne promenljive 4

Funkciju **newGame**() koristimo za inicijalizaciju potrebnih varijabli pre pocetka svake partije.

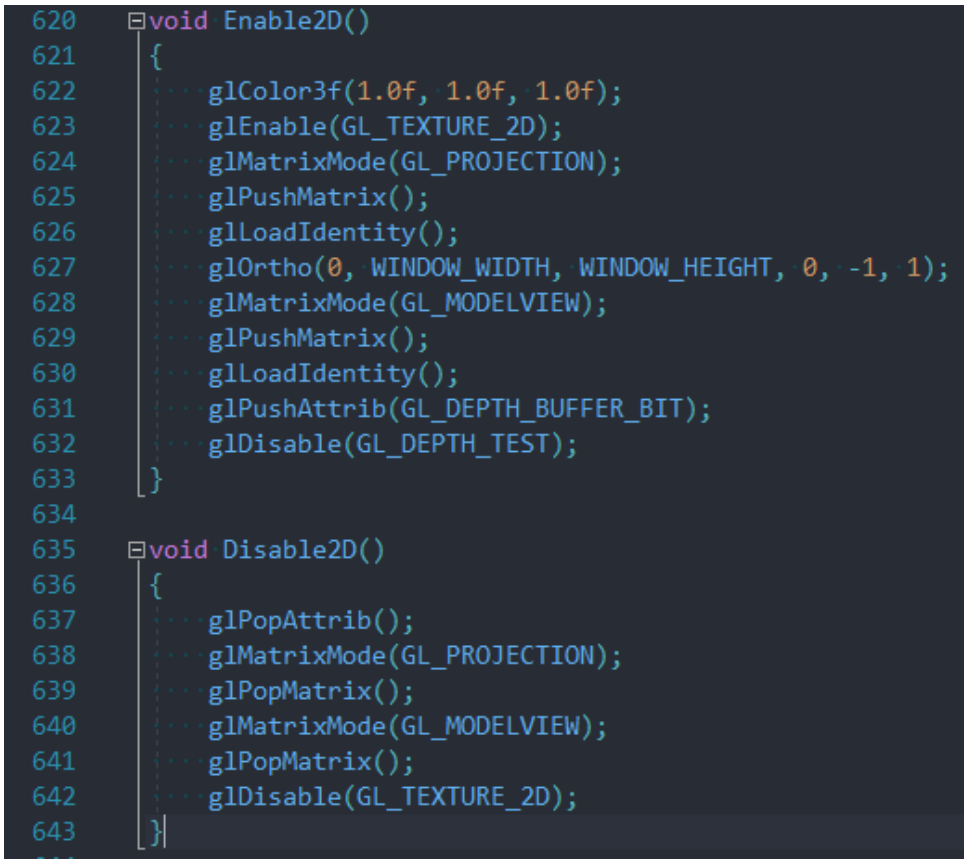
```
1077    void newGame()
1078    {
1079        chess = new Game();
1080        selectedRow = 1; selectedCol = 1;
1081        moveToRow = 1; moveToCol = 1;
1082        selected = false;
1083        board_rotating = true;
1084        rotation = 0;
1085        inGame = true;
1086        gameState = GameState::IN_PROGRESS;
1087        check = false;
1088        checkMate = false;
1089        timeOut = false;
1090        isOverPlayed = false;
1091        isTickingPlayed = false;
1092        goalTime = selectedTime;
1093        startTime = glutGet(GLUT_ELAPSED_TIME);
1094        updateTurn(chess->getTurnColor());
1095    }
```

3.2.7 newGame() funckija

- Iscrtavanje(display)

Za ovaj projekat koristim dva rezima crtanja:

• **Render2D** – za iscrtavanje UI elemenata poput glavnog menija ili teksta tokom igre. Radi na princip ukljucivanja i iskljucivanja rezima za prikaz u 2D prostoru na pocetku i kraju funkcije.



```
void Enable2D()
{
    glColor3f(1.0f, 1.0f, 1.0f);
    glEnable(GL_TEXTURE_2D);
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glLoadIdentity();
    glOrtho(0, WINDOW_WIDTH, WINDOW_HEIGHT, 0, -1, 1);
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();
    glPushAttrib(GL_DEPTH_BUFFER_BIT);
    glDisable(GL_DEPTH_TEST);
}

void Disable2D()
{
    glPopAttrib();
    glMatrixMode(GL_PROJECTION);
    glPopMatrix();
    glMatrixMode(GL_MODELVIEW);
    glPopMatrix();
    glDisable(GL_TEXTURE_2D);
}
```

3.3.1 **Enable2D** i **Disable2D** funkcije

```
647     Enable2D();
648
649     switch (gameState)
650     {
651     case SPLASH:
652         if (elapsedTime >= SPLASH_THRESHOLD)
653             gameState = MENU;
654
655         splashScreen->Render();
656         break;
657
658     case MENU:
659         menuScreen->Render();
660
661         arrowIcon->SetPosition(arrowPositions[arrowIndex]);
662         arrowIcon->Render();
663         break;
664
665     case CONTROLS:
666         controlsScreen->Render();
667         break;
668
669     case PAUSED:
670         isTickingPlayed = false;
671         backgroundChannel->setPaused(true);
672
673         pausedScreen->Render();
674
675         arrowIcon->SetPosition(arrowPositions[arrowIndex]);
676         arrowIcon->Render();
677         break;
678
679     case MODE_SELECTION:
680         modeScreen->Render();
681
682         arrowIcon->SetPosition(arrowPositions[arrowIndex]);
683         arrowIcon->Render();
684         break;
685
686     case IN_PROGRESS:
687         // starting the ticking sound once, when the game starts
688         if (!isTickingPlayed) {
689             audioMgr->playSound(sfxTicking, 0, false, &backgroundChannel);
690             backgroundChannel->setPaused(false);
691             backgroundChannel->setLoopCount(-1);
692             backgroundChannel->setVolume(1);
693
694             isTickingPlayed = true;
695         }
696         // stoping the sound when the game ends
697         if (timeOut || checkMate && isTickingPlayed) {
698             backgroundChannel->setPaused(true);
699             isTickingPlayed = false;
700         }
701
702         backgroundScreen->Render();
703         break;
704     }
705     Disable2D();
```

3.3.2 Implementacija **Render2D** funkcije(iscrtavanje UI-a)

• **Display** – za iscrtavanje svega; ovde zovemo funkciju Render2D() pre iscrtavanja svih elemenata igre kako bi se oni nacrtali jedni preko drugih. Nakon toga se vracamo u prikaz 3D

formata gde menjamo perskeptivu i model view matrice, inicijalizujemo i prikazujemo osvetljenje, zovemo sve funkcije za iscrtavanje same sahovske table i njenih figura poput **drawChessPieces()** i **drawBoardSquares()** i na kraju prikazujemo UI text po potrebi.

```
467  void drawChessPieces()
468  {
469      float z;
470      for (int row = 1; row <= 8; row++)
471      {
472          for (int col = 1; col <= 8; col++)
473          {
474              if (chess->isSquareOccupied(row, col))
475              {
476                  glPushMatrix();
477                  if (selected && row == selectedRow && col == selectedCol) z = 1.0;
478                  else z = 0.6;
479                  glTranslatef((row - 5) * 1.0f + 0.5f, (col - 5) * 1.0f + 0.5f, z);
480                  glScalef(.18f, .18f, .18f);
481                  switch (chess->getPieceColor(row, col))
482                  {
483                  case PieceColor::WHITE:
484                      glRotatef(90, 0.0f, 0.0f, 1.0f);
485                      glColor3f(0.9f, 0.9f, 0.9f);
486                      break;
487                  case PieceColor::BLACK:
488                      glRotatef(-90, 0.0f, 0.0f, 1.0f);
489                      glColor3f(0.1f, 0.1f, 0.1f);
490                      break;
491                  }
492                  switch (chess->getPiece(row, col)->getType())
493                  {
494                      case PieceType::PAWN: Pawn.Draw(); break;
495                      case PieceType::ROOK: Rook.Draw(); break;
496                      case PieceType::KNIGHT: Knight.Draw(); break;
497                      case PieceType::BISHOP: Bishop.Draw(); break;
498                      case PieceType::QUEEN: Queen.Draw(); break;
499                      case PieceType::KING: King.Draw(); break;
500                  }
501                  glPopMatrix();
502              }
503          }
504      }
505      glColor3f(0, 0, 0);
506  }
```

3.3.3 **drawChessPieces**() funkcija

```
void drawBoardSquares()
{
    float r, c;
    for (int row = 1; row <= 8; row++)
    {
        for (int col = 1; col <= 8; col++)
        {
            r = 1.0 * (row - 5);
            c = 1.0 * (col - 5);
            if (row == selectedRow && col == selectedCol)
            {
                if (selected) glColor3f(0.33f, 0.420f, 0.184f);
                else if (chess->isSquareOccupied(selectedRow, selectedCol))
                    if (chess->getPieceColor(selectedRow, selectedCol) == chess->getTurnColor())
                        glColor3f(0.0f, 0.5f, 0.0f);
                    else glColor3f(1.0f, 0.0f, 0.0f);
                else glColor3f(0.3f, 0.7f, 0.5f);
            }
            else if ((row + col) & 1) glColor3f(1.0, 1.0, 1.0);
            else glColor3f(0.0, 0.0, 0.0);
            glPushMatrix();
                glTranslatef(r, c, 0.5f);
                glBegin(GL_TRIANGLES);
                    glNormal3fv(normal_board);
                    glVertex3f(0.0f, 0.0f, 0.0f);
                    glVertex3f(1.0f, 1.0f, 0.0f);
                    glVertex3f(0.0f, 1.0f, 0.0f);
                    glVertex3f(0.0f, 0.0f, 0.0f);
                    glVertex3f(1.0f, 1.0f, 0.0f);
                    glVertex3f(1.0f, 0.0f, 0.0f);
                glEnd();
            glPopMatrix();
        }
    }
    glColor3f(0, 0, 0);
}
```

3.3.4 **drawBoardSquares**() funkcija

4. ISPITNI ZADACI

1. Zadatak Implementirati ograničeno vreme za potez. Svaki igrač ima samo neko određeno vreme da odigra svoj potez, kao što je to u profesionalnom šahu. Prikazivati ovo vreme u vidu brojača na ekranu u toku igre.

2. Zadatak Ubaciti nekoliko različitih režima igre, najverovatnije zasnovanih na sistemu vremena (npr. partija sa standardnim vremenskim ograničenjem, brzinska partija sa kraćim potezima, i partija sa neograničenim vremenom). Ukoliko nije uspešno urađen zadatak sa implementiranjem vremena, možda različiti režimi igre mogu da se zasnivaju na broju poteza. Selekciju režima igre ubaciti u glavni meni ili u pod-meni nakon odabira opcije „Start“ u glavnom meniju.

3. Zadatak Implementirati pauzu, sa svojim menijem iz kog je moguće vratiti se u glavni meni ili zatvoriti program igre. Pauza, naravno, zaustavlja odbrojavanje vremena za potez (ako je implementirano) i oduzima igračima kontrolu nad šahovskim figurama.

BONUS Pored brojača za vremensko ograničenje (ako je implementirano), ubaciti druge elemente korisničkog interfejsa u igru, npr. pojedene figurice i indikator igrača čiji je potez trenutno. Pored toga, ubaciti zvučne efekte u igru, gde je to prikladno (postavljanje figurica, kliktanje u meniju, isticanje vremena, itd.)

1. Zadatak

```
753         // Time calculation and printing
754         if (!unlimitedGame) {
755             showTime = (goalTime - (elapsedTime - startTime));
756 
757             if (showTime / 1000 <= 0)
758                 timeOut = true;
759             if(!checkMate && !timeOut)
760                 showWord(-27, WINDOW_HEIGHT / 2 - 70, printTime(showTime / 1000));
761         }
762         if (!checkMate && !timeOut) {
763             std::string s = chess->getTurnColor() == PieceColor::BLACK ? "BLACK's MOVE" : "WHITE's MOVE";
764             showWord(-92, WINDOW_HEIGHT / 2 - 50, s);
765         }
766 
```

4.1.1 Kod koriscen za resavanje ovog zadatka

showTime – preostalo vreme za pokret (vreme koje se prikazuje na korisnickom interfejsu)

goalTime – ukupno vreme cele igre

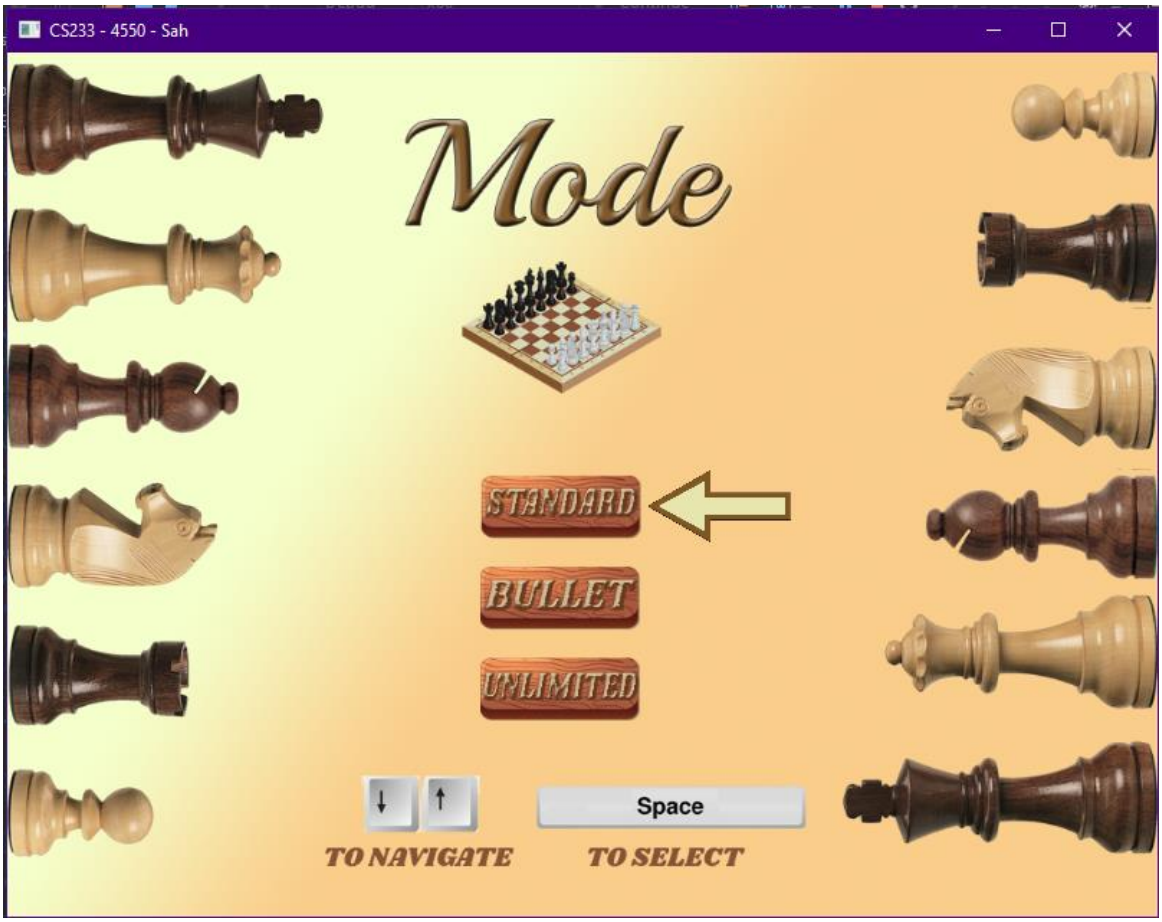
elapsedTime – ukupno proteklo vreme od pocetka programa koje uzimamo sa glutGet(GLUT_ELAPSED_TIME) u svakom frejmu

startTime – vreme kad je partija saha zapoceta

```
151 /**
152     Returns full time from seconds, from 126 to 02:06
153 
154 BUG: doesnt work for times over 10 minutes, it will write "012:15"
155     redundant to implement that here
156 */
157 std::string printTime(int time)
158 {
159     if (time < 60)
160         return std::to_string(time) + 's';
161 
162     int minutes = time / 60;
163     int seconds = time % 60;
164 
165     // returns the time formated properly, if seconds < 10 we add a 0 before the seconds to avoid ->(02:5)
166     return seconds < 10 ?
167         '0' + std::to_string(minutes) + ':' + '0' + std::to_string(seconds) :
168         '0' + std::to_string(minutes) + ':' + std::to_string(seconds);
169 }
```

4.1.2 printTime(int time) funkcija

2. Zadatak



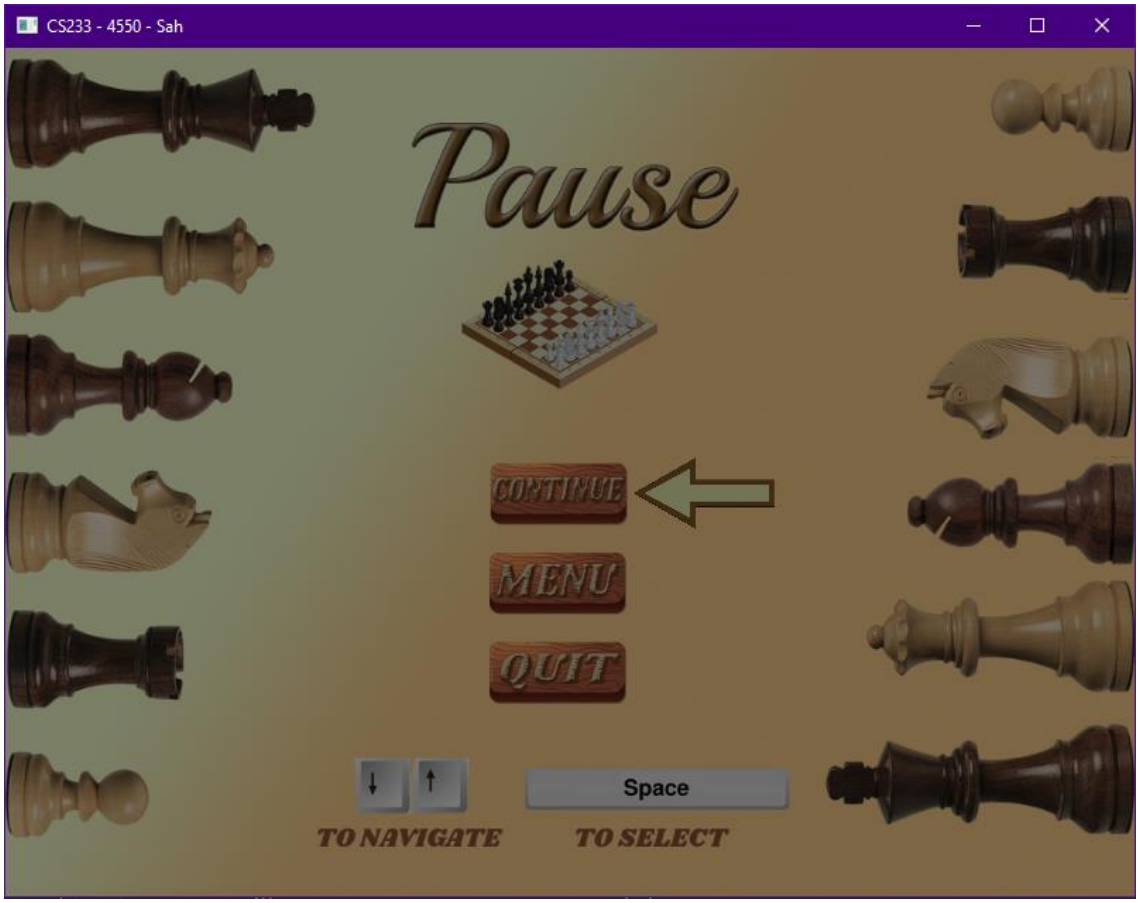
4.2.1 Drugi zadatak (selekcija moda)

Ovaj zadatak sam resio tako sto sam **keyboard** funkciji svaki put kad korisnik klikne space proveravao da li je **gameState** u stanje **MODE_SELECTION,** ukoliko jeste onda u zavisnosti od izabrane opcija definisemo **unlimitedTime** i **unlimitedGame** promenljive koje odredjuju duzinu igre.

```
929         else if (gameState == GameState::MODE_SELECTION) {
930             audioMgr->playSound(sfxClick, 0, false, &sfxChannel);
931             sfxChannel->setVolume(3);
932 
933             switch (arrowIndex) {
934             case 0:
935                 unlimitedGame = true;
936                 break;
937             case 1:
938                 //selectedTime = 1000; for testing(timer = 1sec)
939                 selectedTime = 90000;
940                 unlimitedGame = false;
941                 break;
942             case 2:
943                 selectedTime = 300000;
944                 unlimitedGame = false;
945                 break;
946             }
947             newGame();
948         }
```

4.2.2 Implementacija drugog zadatka

3. Zadatak



4.3.1 Treci zadatak (pauza)

```
1033        case 'p': case 'P':
1034            if (gameState == GameState::PAUSED) {
1035                // Compensating for time lost during pause
1036                unPausedTime = glutGet(GLUT_ELAPSED_TIME);
1037                goalTime += (unPausedTime - pausedTime) / 2;
1038                startTime += (unPausedTime - pausedTime) / 2;
1039                gameState = GameState::IN_PROGRESS;
1040                return;
1041            }
1042
1043            // Restricting pauses outside of in_progress mode
1044            if (gameState != GameState::IN_PROGRESS)
1045                return;
1046
1047            pausedTime = glutGet(GLUT_ELAPSED_TIME);
1048            gameState = GameState::PAUSED;
1049            break;
```

4.3.2 Implementacija treceg zadatka

4. BONUS

- Zvukovi

Zvukove sam implementirao koristeci FMOD biblioteku. Na pocetku programa inicijalizujem sam FMOD i zvuke iz .wav fajlova, i onda ih po potrebi pustam.

Zvukove koje sam dodao su: zvucni efekat za svaki ‘select’ klik u meniju, zvucni efekat pri kraju igre i pozadinski zvuk sata koji otkucava koji je pusten tokom partije saha.

```
171  bool initFmod()
172  {
173      FMOD_RESULT result;
174      result = FMOD::System_Create(&audioMgr);
175      if (result != FMOD_OK)
176      {
177          return false;
178      }
179      result = audioMgr->init(50, FMOD_INIT_NORMAL, NULL);
180      if (result != FMOD_OK)
181      {
182          return false;
183      }
184      return true;
185  }
186
187  const bool loadAudio()
188  {
189      FMOD_RESULT result;
190
191      result = audioMgr->createSound("Sounds/clockTicking.wav", FMOD_LOOP_NORMAL, 0, &sfxTicking);
192
193      result = audioMgr->createSound("Sounds/gameOver.wav", FMOD_DEFAULT, 0, &sfxOver);
194      result = audioMgr->createSound("Sounds/wood.wav", FMOD_DEFAULT, 0, &sfxClick);
195
196      return true;
197  }
```

4.4.1 Inicijalizacija FMOD-a i zvukova za BONUS

- ## Dodatan korisnciki interfejs

Za dodavanje dodatnih sprajtova pojedenih figurica nazalost nisam imao vremena, ali sam ispisivanje indikatora igraca cije je potez trenutno prikazao u slici 4.1.1 unutar drugog glavnog if-a.

5. LITERATURA

- CS233 PREDAVANJA I VEZBE
- Chess Programming
- Chess wiki