Универзитет у Крагујевцу
Факултет инжењерских наука

Рачунарска графика

Семинарски рад

OpenGL – Basketball (1979)

Професор:
Проф. др Ненад Филиповић

Студент:
Стефановић Филип 655/2019

__.__. 2023.

Fakultet inženjerskih nauka
Univerzitet u Kragujevcu

Računarska tehnika i softversko inženjerstvo
IV godina
VIII semestar

Predmet: Računarska grafika

SEMINARSKI RAD IZ PREDMETA RAČUNARSKA GRAFIKA

1. Napraviti aplikaciju igrice Basketball koristeći OpenGL. Opis igrice nalazi se na linku https://en.wikipedia.org/wiki/Basketball_(1979_video_game)
   Omogućiti različite opcije (skupljanje poena, skupljanje/gubljenje života, bar dva nivoa itd.). Preporučuje se razvojno okruženje programskog jezika Visual C++ 6.0 ili Visual Studio (verzije sve do 2015).

Ime i prezime studenta, broj indeksa:

1. _____

Seminarski rad je potrebno poslati na tijanas@kg.ac.rs (izvorni kod programa, izvršna verzija, pomoćne datoteke itd.) i predati u pisanom obliku.

Pisani deo seminarskog rada povezan u spiralu treba da sadrži sledeće delove:
- naslovnu strana sa jasno napisanim osnovnim podacima
- postavku zadatka
- opis delova programa sa ilustracijama i samim izvornim kodom
- spisak korišćene literature

Odbrana seminarskog rada je deo usmenog ispita koji se brani u zakazanom terminu. Kandidat je u obavezi da praktičnom demonstracijom rada programa detaljno objasni rad pojedinih delova programa.

Kragujevac,
29. mart, 2023. godine

PREDMETNI NASTAVNIK
dr Nenad Filipović, red. prof.

Садржај

1 Увод

Basketball је видео игра која је објављена 1979. године за Атари 2600 конзолу. Игра је једноставна спортска симулација кошарке. Будући да су могућности и графичке способности тог времена биле ограничене, игра је пружала основно искуство играња кошарке.

У игри Basketball, играч има контролу над једним играчем на терену. Циљ је постићи што више поена тако што ћете убацити лопту у противнички кош.

Ево основних правила игре:

1. Играчи: Игра се може играти против рачунара или против другог играча. Сваки играч има контролу над једним играчем на терену.

2. Контроле: Играч користи џојстик за кретање свог играча по терену. Једно дугме на џојстику се користи за шутирање лопте према кошу, а друго за крађу лопте од противника.

3. Кретање играча: Играч може кретати свог играча напред, назад, лево и десно по терену, и може скакати.

4. Шутирање: Да бисте шутирали лопту према кошу, играч притиска дугме на џојстику. Прецизност шута зависи од позиције и тренутне удаљености играча од коша.

5. Противници: Противнички играчи такође се крећу по терену и покушавају спречити играча да постигне поене.

6. Бодовање: Сваки пут када играч убаци лопту у противнички кош, осваја један поен.

Игра Basketball из 1979. године није имала комплексне елементе као модерније спортске симулације, али је пружала забавно искуство играња кошарке играчима тога времена.

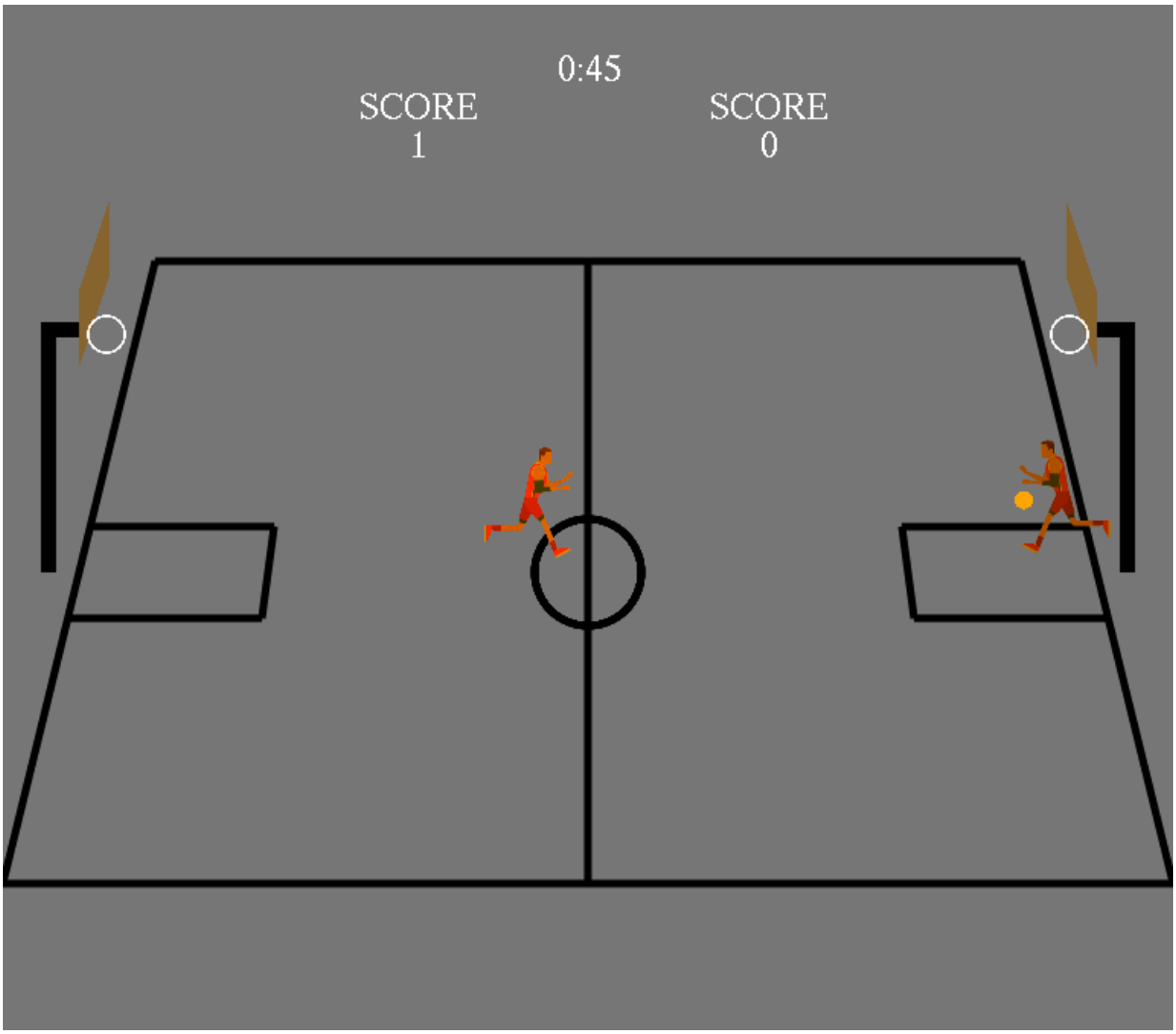
2 Игра

У оквиру ове секције продискутоваће се дизајн корисничког интерфејса, корисничка ограничења и могућности. Игра је раздвојена у две сцене, свет у којем се игра дешава и мали мени који представља завршетак игре.

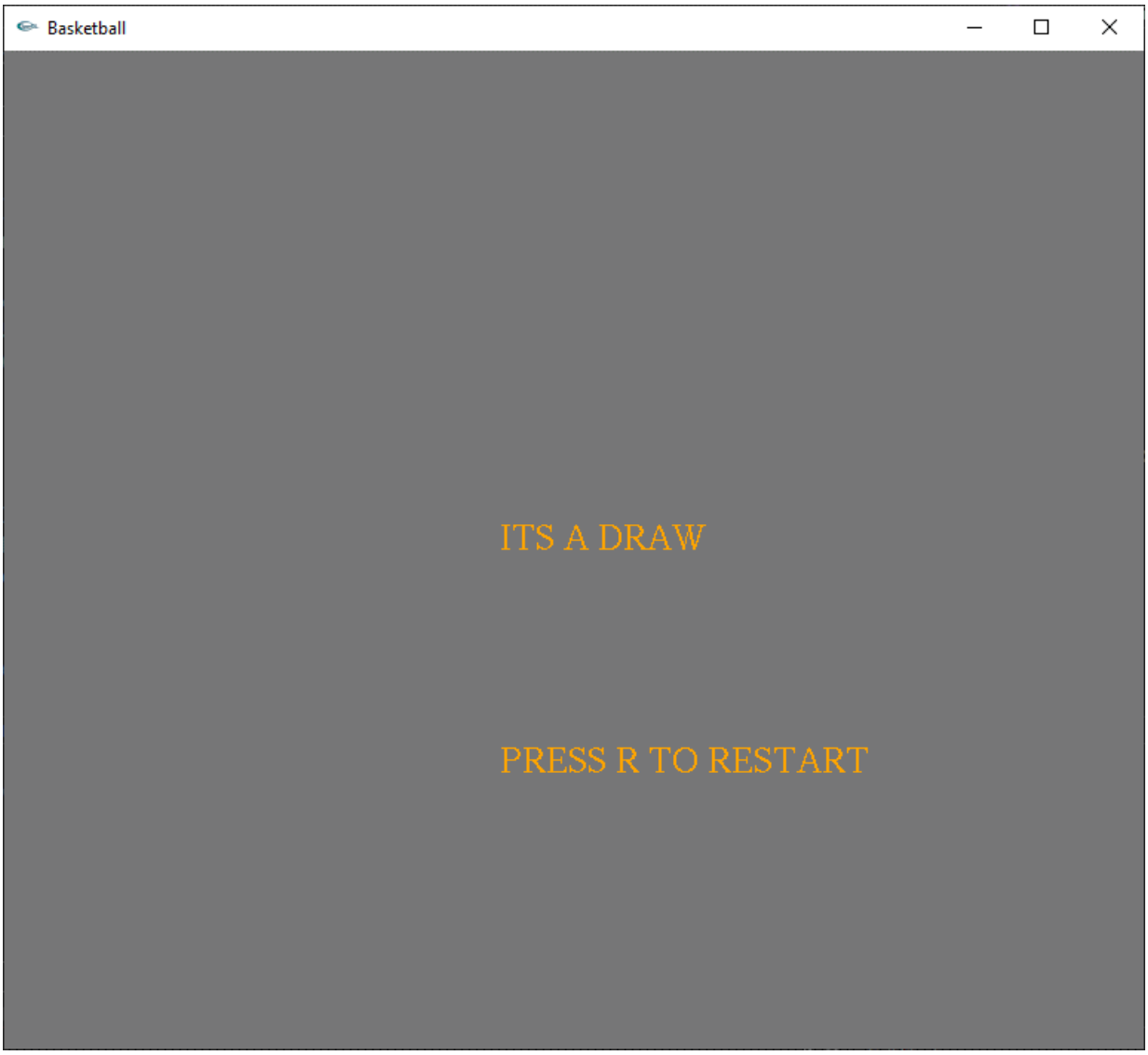
2.1 Свет игре

Приликом покретања игре појављује се нова сцена на прозору. Ова сцена представља дводимензионалну раван на чијим осама се шетају играчи.
На почетку, лопта је у средини и играчи јуре да је покупе, након чега заправо креће игра. Након сваког постигнутог поена врши се ресет. Играч који је постигао поен се враћа на његову страну терена ка центру, док се други играч враћа испод његовог коша и добија посед лопте. Играч може само шутирати уколико има лопту, док може скакати и покушавати отимање лопте уколико је нема. Када истекне време, игра се прекида и проглашава се победник.
Сликама 1, 2, и 3 приказан је свет игре и стања у којима може бити.

Слика 1: Почетно стање



Слика 2: Постигнут кош



Слика 3: Крај игре

3 Анализа кода и логике

Ова апликација или игра зависи од следећих екстерних библиотека:

- stb_image
  Ова библиотека омогућава апликацији да лако учитава текстуре. Методама класе која је дефинисана у оквиру библиотеке се учитава текстура. На овом линку је могуће преузети библиотеку.

  https://github.com/nothings/stb/blob/master/stb_image.h

- Glut
  Библиотека за исцртавање по прозору. Омогућава креирање потребног интерфејса игре.

Апликација се састоји из дела програма који црта елементе игре на екрану, главне петље у коме се обрађује логика игрице, услужне класе Player, структуре Movement, и броја глобалних промењљивих.

3.1 Класа Player

Класа Player представља играча.

```
class Player {
private:
    float posX;
    float posY;
    bool flipped;
    bool jumping;
    float jumpStartTime;
    float jumpStartY;
    bool shootKeyHeld;
    bool ballPossesion;
    float shootStrength;
```

Сви атрибути класе Player су приватни. Опис атрибута дат је наредном табелом.

posX	X координата играча
posY	координата играча
flipped	Логичка променљива - да ли је играч окренут
jumping	Логичка променљива - да ли играч скаче
jumpStartTime	Време у ком је играч започео скок
jumpStartY	Координата на којој је играч започео скок
shootKeyHeld	Логичка променљива – да ли играч држи дугме за шутирање
ballPossesion	Логичка променљива – да ли играч има посед лопте
shootStrength	Снага са којом је играч избацио лопту

```
public:
    Player(float initialX, float initialY, bool flipped)
        : posX(initialX), posY(initialY), flipped(flipped), jumping(false),
        jumpStartTime(0.0), jumpStartY(0.0), shootKeyHeld(false),
ballPossesion(false), shootStrength(0.0) {}

    float getPositionX() const { return posX; }
    float getPositionY() const { return posY; }
    bool isFlipped() const { return flipped; }
    bool isJumping() const { return jumping; }
    float getJumpStartTime() const { return jumpStartTime; }
    float getJumpStartY() const { return jumpStartY; }
    bool getShootKeyHeld() const { return shootKeyHeld; }
    bool getBallPossesion() const { return ballPossesion; }
    float getShootStrength() const { return shootStrength; }

    void setPositionX(float newX) { posX = newX; }
    void setPositionY(float newY) { posY = newY; }
    void setFlipped(bool isFlipped) { flipped = isFlipped; }
    void setJumping(bool isJumping) { jumping = isJumping; }
    void setJumpStartTime(float time) { jumpStartTime = time; }
    void setJumpStartY(float startY) { jumpStartY = startY; }
    void setShootKeyHeld(bool shootHeld) { shootKeyHeld = shootHeld; }
    void setBallPossesion(bool possesion) { ballPossesion = possesion; }
    void setShootStrength(float strength) { shootStrength = strength; }
```

Све методе класе су јавне, као и конструктор. Конструктор на основу координата центра играча и да ли треба да буде флипован креира објекат са прослеђеним параметрима.

```
void move(float deltaX, float deltaY) {
        posX += deltaX;
        posY += deltaY;
    }
```

Метода `move помера позицију играча према задатим параметрима.`

```
void shootBall(float shotStrength) {
        if(ballPossesion) {
            if (!shotFired) {
                initialBallY = ballY;
                float minSpeed = 1.0;
                float maxSpeed = 15.0;
                float minAngle = 45.0;
                float maxAngle = 75.0;

                float shotSpeed = minSpeed + (shotStrength * (maxSpeed - minSpeed));
                shotSpeed = fminf(fmaxf(shotSpeed, minSpeed), maxSpeed);

                float shotAngle = minAngle + (shotStrength * (maxAngle - minAngle));
                shotAngle = fminf(fmaxf(shotAngle, minAngle), maxAngle);

                float radians = shotAngle * M_PI / 180.0;
                if (!flipped) {
                    ballSpeedX = fabsf(shotSpeed * cosf(radians));
                }
                else {
                    ballSpeedX = -fabsf(shotSpeed * cosf(radians));
                }
                ballSpeedY = fabsf(shotSpeed * sinf(radians));
                shotFired = true;
                ballPossesion = false;
                shootStrength = 0.0;
            }
        }
```

Метода shootBall шутира лопту играча након што се пусти дугме за шутирање. Поставља се иницијална Y координата лопте. Рачуна се угао и снага шута у зависности колико дуго је држано дугме. Ако је играч флипован брзина је негатвина.

3.2 Цртање сцене

```
void drawScene() {
    if (!endGame) {
        glClearColor(0.4609375, 0.4609375, 0.46484375, 0.0);
        glClear(GL_COLOR_BUFFER_BIT);

        drawTime();
        drawScore();
        drawStaticElements();
        drawBall(ballX, ballY + jump_offset);
        drawPlayer(player1Texture, player1.getPositionX(), player1.getPositionY(),
player1.isFlipped());
        drawPlayer(player2Texture, player2.getPositionX(), player2.getPositionY(),
player2.isFlipped());
    }

    if (endGame) {
        glClear(GL_COLOR_BUFFER_BIT);
        if (scoreP1 > scoreP2) {
            drawText("Player 1 WON", WINDOW_WIDTH / 2  - 50,WINDOW_HEIGHT / 2);
        }
        else if(scoreP1 < scoreP2) {
            drawText("Player 2 WON", WINDOW_WIDTH / 2 - 50, WINDOW_HEIGHT / 2);
        }
        else {
            drawText("ITS A DRAW", WINDOW_WIDTH / 2 - 50, WINDOW_HEIGHT / 2);
        }
        drawText("PRESS R TO RESTART", WINDOW_WIDTH / 2 - 50, WINDOW_HEIGHT / 2 -
150);
    }

    glutSwapBuffers();

}
```

Цртање сцене се састоји из подешавања боје позадине, цртања преосталог времена, цртања резултата, цртања статичких елемената, цртања лопте, и цртања играча у случају да се игра није завршила, иначе се црта екран завршетка игре.

3.2.1 Цртање статичких елемената

```
void drawStaticElements() {
    // Draw the court rectangle
    glColor3fv(colors[BLACK]);
    glLineWidth(5.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(0.0, courtdownY);
    glVertex2f(WINDOW_WIDTH, courtdownY);
    glVertex2f(WINDOW_WIDTH - 100, courtupY);
    glVertex2f(100, courtupY);
    glEnd();

    // Draw the court center line
    glBegin(GL_LINES);
    glVertex2f(WINDOW_WIDTH / 2, courtupY);
    glVertex2f(WINDOW_WIDTH / 2, courtdownY);
    glEnd();

    // Draw the threepoint square left side
    glBegin(GL_LINE_STRIP);
    glVertex2f(58, (courtdownY + courtupY) / 2 + 30);
    glVertex2f(178, (courtdownY + courtupY) / 2 + 30);
    glVertex2f(170, (courtdownY + courtupY) / 2 - 30);
    glVertex2f(43, (courtdownY + courtupY) / 2 - 30);
    glEnd();

    // Draw the threepoint square right side
    glBegin(GL_LINE_STRIP);
    glVertex2f(WINDOW_WIDTH - 58, (courtdownY + courtupY) / 2 + 30);
    glVertex2f(WINDOW_WIDTH - 178, (courtdownY + courtupY) / 2 + 30);
    glVertex2f(WINDOW_WIDTH - 170, (courtdownY + courtupY) / 2 - 30);
    glVertex2f(WINDOW_WIDTH - 43, (courtdownY + courtupY) / 2 - 30);
    glEnd();


    // Draw the center line circle
    drawCircle(GL_LINE_LOOP, WINDOW_WIDTH / 2, (courtdownY + courtupY) / 2, 35);
```

Овде се црта терен са централном линијом, централним кругом и квадратима за три поена.

```
//Hoop pole 1
glBegin(GL_QUADS);
glVertex2f(25, (courtdownY + courtupY) / 2);
glVertex2f(25, (courtdownY + courtupY) / 2 + pole_height);
glVertex2f(35, (courtdownY + courtupY) / 2 + pole_height);
glVertex2f(35, (courtdownY + courtupY) / 2);
glEnd();
//Hoop pole 2
glBegin(GL_QUADS);
glVertex2f(25, courtupY - 50);
glVertex2f(25, courtupY - 40);
glVertex2f(50, courtupY - 40);
glVertex2f(50, courtupY - 50);
glEnd();

//Hoop backboard
glColor3fv(colors[HOOP_BACKBOARD]);
glBegin(GL_QUADS);
glVertex2f(50, courtupY - 70);
glVertex2f(50, courtupY - 20);
glVertex2f(70, courtupY + 40);
glVertex2f(70, courtupY - 10);
glEnd();

//Hoop rim
glLineWidth(1.0);
glColor3fv(colors[WHITE]);
drawCircle(GL_LINE_LOOP, hoopLeftRimX, hoopRimY, hoopRimRadius);
```

Овде се црта леви кош и сви његови елементи. Десни кош се на исти начин црта само са другим координатама.

3.2.2 Цртање резултата

```
void drawScore() {
    glColor3fv(colors[WHITE]);

    std::string score = "SCORE";
    float player1X = WINDOW_WIDTH / 2 - 150;
    float player1Y = WINDOW_HEIGHT - 75;
    drawText(score, player1X, player1Y);

    std::string str_scoreP1 = std::to_string(scoreP1);
    float scoreP1X = WINDOW_WIDTH / 2 - 117;
    float scoreP1Y = WINDOW_HEIGHT - 100;
    drawText(str_scoreP1, scoreP1X, scoreP1Y);

    float player2X = WINDOW_WIDTH / 2 + 80;
    float player2Y = WINDOW_HEIGHT - 75;
    drawText(score, player2X, player2Y);

    std::string str_scoreP2 = std::to_string(scoreP2);
    float scoreP2X = WINDOW_WIDTH / 2 + 113;
    float scoreP2Y = WINDOW_HEIGHT - 100;
    drawText(str_scoreP2, scoreP2X, scoreP2Y);
}
```

Овде се цртају резултати играча, дефинишу се координате а функцијом drawText се врши само цртање.

```
void drawText(const std::string& text, float x, float y) {
    for (char c : text) {
        glRasterPos2f(x, y);
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, c);
        x += glutBitmapWidth(GLUT_BITMAP_TIMES_ROMAN_24, c);
    }
}
```

Пролази се кроз сваки карактер у тексту, дефинише се позиција, црта се користећи glutBitmapCharacter функцију и онда се повећава x за дужину карактера.

3.2.3 Цртање лопте

```
void drawBall(float x, float y) {
    glColor3fv(colors[ORANGE]);
    drawCircle(GL_POLYGON, x, y, ballRadius);
}
```

Дефинише се боја лопте, добијају се x и y параметри и коришћењем методе drawCircle црта се лопта.

```
void drawCircle(int GLprimitive, float centerX, float centerY, float radius) {
    glBegin(GLprimitive);
    for (int i = 0; i < 1080; ++i) {
        float theta = i * M_PI / 180.0;
        float x = radius * cos(theta);
        float y = radius * sin(theta);
        glVertex2f(x + centerX, y + centerY);
    }
    glEnd();
}
```

Функција добија примитив који ће се користити за цртање, добија x и y центра лопте и добија полупречник лопте.

3.2.4 Цртање играча

```
void drawPlayer(GLuint playerTexture, float x, float y, bool flipped) {
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, playerTexture);
    glBegin(GL_QUADS);
    glTexCoord2f(flipped, 1.0);
    glVertex2f(x, playerHeight + y);
    glTexCoord2f(flipped, 0.0);
    glVertex2f(x, y);
    glTexCoord2f(!flipped, 0.0);
    glVertex2f(playerWidth + x, y);
    glTexCoord2f(!flipped, 1.0);
    glVertex2f(playerWidth + x, playerHeight + y);
    glEnd();
    glDisable(GL_TEXTURE_2D);
}
```

Функција добија ид ткстуре која ће се користити за исцртавање, x и y играча и да ли је флипован.

3.2.5 Цртање времена

```
void drawTime() {
    glColor3fv(colors[WHITE]);
    if (remainingMinutes < 0) {
        endGame = true;
    }
    std::string timerText = std::to_string(remainingMinutes) + ":" +
(remainingSeconds < 10 ? "0" : "") + std::to_string(remainingSeconds);
    drawText(timerText, WINDOW_WIDTH / 2 - 20, WINDOW_HEIGHT - 50);
}
```

Функција генерише текст који представља време и исцртава га drawText функцијом.

3.3 Логика игре

```
void timer(int value) {
    if(!endGame)
    {
        if (player1.getShootKeyHeld() && !shotFired) {
            player1.setShootStrength(player1.getShootStrength() + 0.035);
        }
        if (player2.getShootKeyHeld() && !shotFired) {
            player2.setShootStrength(player2.getShootStrength() + 0.035);
        }
        handleShooting(ballX, ballY, ballSpeedX, ballSpeedY, shotFired);
        handleJump(player1, jumpDuration, jumpHeight);
        handleJump(player2, jumpDuration, jumpHeight);
        checkBallCollision();
        handlePlayerBallCollision(ballX, ballY);
        screenCollision();
        checkIfScored();
        if (!((player1.getBallPossesion() && player1.getShootKeyHeld()) ||
(player2.getBallPossesion() && player2.getShootKeyHeld()))) {
            dribbleBall();
        }

        time_ball += TIME_INCREMENT;
    }
    handleInput();
    calculateTimeRemaining();
    glutPostRedisplay();
    glutTimerFunc(16, timer, 0);
}
```

Сва логика игре се дешава у timer функцији која се позива сваких 16мс што представља око 60 фпс-а. Ако се игра није завршила онда се рачуна јачина шута играча, и онда се редом позивају функције за проверу и извршавање разних догађаја.

3.3.1 handleShooting()

```
void handleShooting(float& ballX, float& ballY, float& ballSpeedX, float&
ballSpeedY, bool& shotFired) {
    if (shotFired) {
        ballY += ballSpeedY;
        ballX += ballSpeedX;
        ballSpeedY -= gravity;

        if (initialBallY - ballY >= pole_height) {
            ballSpeedX = 0.0f;
            ballSpeedY = 0.0f;
            shotFired = false;
        }
    }
}
```

Ова функција узима тренутну позицију лопте, њену израчунату брзину и да ли је испаљена. Рачуна нове координате лопте у зависности од њене брзине, даје гравитацију лопти. Ако лопта падне за висину коша испод своје иницијалне Y координате зауставља се.

3.3.2 handleJump()

```
void handleJump(Player& player, float jumpDuration, float jumpHeight) {
    if (player.isJumping()) {
        float elapsedTime = static_cast<float>(clock() - player.getJumpStartTime())
/ CLOCKS_PER_SEC;
        if (elapsedTime <= jumpDuration) {
            float t = elapsedTime / jumpDuration;
            float jumpOffset = jumpHeight * 4.0 * t * (1.0 - t);
            float newY = player.getJumpStartY() + jumpOffset;
            player.setPositionY(newY);
        }
        else {
            player.setJumping(false);
        }
    }
}
```

Ова функција узима играча, глобалне промењљиве дужина скока и висина скока. Ако је играч у скоку онда се рачуна протекло време скока. Уколико је време мање или једнако трајању скока, израчунава се нормализовано време (t) као однос протеклог времена и трајања скока. Коришћењем овог нормализованог времена, израчунава се помак скока (jumpOffset) који зависи од висине скока и нормализованог времена. Ново Y координата играча (newY) се добија додавањем помака скока на почетну Y координату скока. Иначе се скок завршио и поставља се setJumping поље играча на false.

3.3.3 checkBallCollision()

```
void checkBallCollision() {
    // Check collision with the backboard
    if (ballY >= courtupY - 70 && ballY <= courtupY + 40) {
        if ((ballX >= 50 && ballX <= 70) || (ballX >= (WINDOW_WIDTH - 70) && ballX
<= (WINDOW_WIDTH - 50))) {
            ballSpeedX *= -0.5;
            ballSpeedY *= -0.5;
        }
    }

    // Check collision with the rim
    float rimDistanceX = fabs(ballX - hoopLeftRimX);
    float rimDistanceY = fabs(ballY - hoopRimY);
    float distanceToRim = sqrt(pow(rimDistanceX, 2) + pow(rimDistanceY, 2));

    if (distanceToRim <= hoopRimRadius + ballRadius) {
        // Ball collided with the rim
        ballSpeedX *= -0.7;
        ballSpeedY *= -0.7;
    }
}
```

Функција која провера колизију лопте са таблом коша и са ободом коша и одбија је.

3.3.4 handlePlayerBallCollision()

```
void handlePlayerBallCollision(float& ballX, float& ballY) {
    touchedP1 = checkPlayerBallCollision(player1.getPositionX(),
player1.getPositionY(), ballX, ballY, playerWidth, playerHeight, ballRadius,
player1.isFlipped());
    touchedP2 = checkPlayerBallCollision(player2.getPositionX(),
player2.getPositionY(), ballX, ballY, playerWidth, playerHeight, ballRadius,
player2.isFlipped());
    //printf("P1: %d, P2: %d\n", touchedP1, touchedP2);

    if (!player1.getBallPossesion() && !player2.getBallPossesion() && touchedP1) {
        player1.setBallPossesion(true);
        player2.setBallPossesion(false);
        shotFired = false;
    }
    else if (!player1.getBallPossesion() && !player2.getBallPossesion() &&
touchedP2) {
        player1.setBallPossesion(false);
        player2.setBallPossesion(true);
        shotFired = false;
    }
    else if (player1.getBallPossesion() && player2StealButtonPressed && touchedP2) {
        player1.setBallPossesion(false);
        player2.setBallPossesion(true);
        shotFired = false;
    }
    else if (player2.getBallPossesion() && player1StealButtonPressed && touchedP1) {
        player1.setBallPossesion(true);
        player2.setBallPossesion(false);
        shotFired = false;
    }

    if (player1.getBallPossesion()) {
        ballY = player1.getPositionY() + 40;
        ballX = player1.getPositionX() + (player1.isFlipped() ? 50 : 70);
    }
    if (player2.getBallPossesion()) {
        ballY = player2.getPositionY() + 40;
        ballX = player2.getPositionX() + (player2.isFlipped() ? 50 : 70);
    }
}
```

Функција која проверава колизију играча и лопте. Прво се рачуна да ли ико од играча има колизију са лоптом. Након тога ако нико нема посед лопте онај који је први имао колизију ће добити посед. Ако неко од играча већ има посед лопте онда се рачуна да ли је крађа лопте успешна. Успешна је ако је играч који краде довољно близу лопти. На крају ако играч има посед лопте, она га прати.

3.3.5 screenCollision()

```
void screenCollision() {
    if (ballX - ballRadius < 0.0 || ballX + ballRadius > WINDOW_WIDTH) {
        ballSpeedX = -ballSpeedX;
    }

    if (ballY - ballRadius < 0.0 || ballY + ballRadius > WINDOW_HEIGHT) {
        ballSpeedY = -ballSpeedY;
    }
}
```

Проверава колизију лопте са екраном и одбија је.

3.3.6 checkIfScored()

```
void checkIfScored() {
    ballInRange = (ballY - ballRadius < hoopRimY + hoopRimRadius) && (ballY +
ballRadius > hoopRimY - hoopRimRadius);
    if (ballInRange) {
        float distanceToLeftRim = abs(ballX - hoopLeftRimX);
        if (distanceToLeftRim < hoopRimRadius && !ballPassedThrough) {
            scoreP2 += 1;
            resetP2Scored();
            ballPassedThrough = true;
        }

        float distanceToRightRim = abs(ballX - hoopRightRimX);
        if (distanceToRightRim < hoopRimRadius && !ballPassedThrough) {
            scoreP1 += 1;
            resetP1Scored();
            ballPassedThrough = true;
        }
    }

    if (!ballInRange) {
        ballPassedThrough = false;
    }
}
```

Проверава да ли је неко од играча дао кош и ако јесте повећава поене том играчу и ресетује игру.

```
void resetP1Scored() {
    resetPlayer(player1, (WINDOW_WIDTH / 2) - 80, courtCenterY, false, false);
    resetPlayer(player2, hoopRightRimX - 80, courtCenterY, true, true);
}

void resetP2Scored() {
    resetPlayer(player1, hoopLeftRimX + 5, courtCenterY, true, false);
    resetPlayer(player2, (WINDOW_WIDTH / 2) - 15, courtCenterY, false, true);
}
void resetPlayer(Player& currentPlayer, float posX, float posY, bool ballPossession,
bool flipped) {
    currentPlayer.setPositionX(posX);
    currentPlayer.setPositionY(posY);
    currentPlayer.setBallPossesion(ballPossession);
    currentPlayer.setFlipped(flipped);
    shotFired = false;
}
```

3.3.7 dribbleBall()

```
void dribbleBall() {
    if (player1.getBallPossesion() || player2.getBallPossesion()) {
        jump_offset = jumpAmplitude * sinf(2.0 * M_PI * jumpFrequency * time_ball);
    }
    else {
        jump_offset = 0.0;
    }
}
```

Функција dribbleBall() одржава посед лопте симулирајући ударање лопте о под. Функција проверава да ли један од играча (player1 или player2) поседује лопту. Ако је то случај, израчунава се помак скока лопте (jump_offset) на основу амплитуде скока (jumpAmplitude), фреквенције скока (jumpFrequency) и времена лопте (time_ball), користећи синусну функцију.

3.3.8 handleInput()

```
void handleInput() {
      if (keyState[27]) {
        glutDestroyWindow(glutGetWindow());
        exit(0);
    }
    if (endGame) {
        if (keyState['r'] || keyState['R']) {
            restartGame();
        }
    }
    // Player 1 movement
    for (const auto& movement : player1Movements) {
        for (const auto& key : movement.key) {
            if (keyState[key]) {
                player1.move(movement.deltaX, movement.deltaY);
                player1.setFlipped(movement.flipped);
                break;
            }
        }
    }

    // Player 1 shooting
    if (keyState['f'] || keyState['F']) {
        player1.setShootKeyHeld(true);
    }
    // Player 1 shooting release
    if (!keyState['f'] && !keyState['F'] && player1.getShootKeyHeld()) {
        player1.setShootKeyHeld(false);
        if (!shotFired) {
            player1.shootBall(player1.getShootStrength());
        }
    }


    // Player 1 jumping
    if (keyState[32]) {
        if (!player1.getBallPossesion()) {
            if (!player1.isJumping()) {
                player1.setJumping(true);
                player1.setJumpStartTime(static_cast<float>(clock()));
                player1.setJumpStartY(player1.getPositionY());
            }
        }
    }

    // Player 1 ball stealing
    if (keyState['r'] || keyState['R']) {
        player1StealButtonPressed = true;
    }
    else {
        player1StealButtonPressed = false;
    }
```

Функција handleInput() проверава и обрађује унос корисника у зависности од тренутног стања игре. Ако је притиснуто дугме escape прозор се затвара и излази са 0 exit кодом. Ако је игра завршена (endGame је true), функција проверава да ли је притиснуто дугме 'r' или 'R', и ако јесте, покреће рестартовање игре позивом функције restartGame().

Након тога, функција обрађује унос за кретање и акције играча 1. У петљи се пролази кроз различите покрете (movement) дефинисане за играча 1. За сваки покрет, пролази се кроз тастере (key) који активирају тај покрет. Ако је неки од тастера притиснут, играч 1 се помера (deltaX, deltaY), поставља се оријентација (flipped) и петља се прекида.

Затим се проверава унос за шутирање играча 1. Ако је тастер 'f' или 'F' притиснут, поставља се индикатор (shootKeyHeld) да је тастер држан.

Након тога, проверава се пуштање тастера за шутирање играча 1. Ако тастер 'f' или 'F' није притиснут и претходно је тастер био држан, поставља се индикатор да тастер више није држан, и ако није већ извршен шут (shotFired), позива се функција shootBall играча 1 са одговарајућом јачином шута (shootStrength).

Затим се проверава унос за скок играча 1. Ако је тастер са ASCII вредношћу 32 (space) притиснут, проверава се да играч 1 нема посед лопте, да није већ у стању скока и да скок није већ покренут. Ако су испуњени ови услови, поставља се стање скока играча 1 на истинито, бележе се време почетка скока, и бележе се почетне координате за скок.

На крају, проверава се унос за крађу лопте играча 1. Ако је тастер 'r' или 'R' притиснут, поставља се индикатор (player1StealButtonPressed) да је тастер за крађу притиснут, иначе се поставља да није притиснут. Све се исто овако обрађује и важи за другог играча.

3.3.9 calculateTimeRemaining()

```
void calculateTimeRemaining() {
    currentTime = time(NULL);
    elapsedSeconds = difftime(currentTime, startTime);
    remainingMinutes = countdownMinutes - 1 - static_cast<int>(elapsedSeconds) / 60;
    remainingSeconds = 59 - static_cast<int>(elapsedSeconds) % 60;
    if (remainingMinutes < 0) {
        endGame = true;
    }
}
```

Рачуна колико је времена остало до краја игре, и ако је прошло поставља проемнљиву endGame на true.

4. Литература

[1] http://moodle.fink.rs, kurs Računarska grafika
[2] https://www.youtube.com/watch?v=r7B05s7sw6g - Видео снимак једног меча
[3] https://ernesthuntley.wordpress.com/2010/08/20/smooth-keyboard-input-with-glut/
[4] https://github.com/nothings/stb