Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Рачунарска техника и софтверско инжењерство
Ниво студија: Основне академске студије
Предмет: Рачунарска графика
Број индекса: 551/2015

Александра Д. Радовић

Развој игрице Билијар у OpenGL окружењу

дипломски рад

Комисија за преглед и одбрану:

1. **Др Ненад Филиповић, ред. проф. - ментор**
2. _____ Датум одбране: _____
3. _____ Оцена: _____

У оквиру овог дипломског рада кандидат треба да направи игрицу Билијар употребом OpenGL програмског интерфејса. Потребно је реализовати графички приказ билијарског стола са свим елементима, билијарских куглица одређених боја и штапа. Постављање куглица у почетни положај треба имплементирати тако да буде у складу са правилима билијара. Потребно је предвидети понашање у одређеним ситуацијама, нпр. када бела куглица прво удари у противникову куглицу, када играч убаци противникову куглицу, када играч убаци црну куглицу, када играч убаци белу куглицу и слично.

Сложеност овог проблема се огледа у одређивању и програмирању параметара након судара две куглице или када куглица удари у ивицу стола. Параметри које је потребно одредити након судара су правац, смер, дужина и брзина кретања куглице. Кориснику апликације треба омогућити управљање штапом помоћу миша и увид у освојене поене у току игрице.

Препоручена литература:

[1] Јаничић, П. (2014). *Рачунарска графика.* Београд: Математички факултет.

[2] Чупић, М.,&Михајловић, Ж. (2014). *Интерактивна рачунална графика кроз примјере у OpenGL-у.*

[3] Којић, М. (1985). *Динамика – теорија и примери.* Београд: Научна књига

Крагујевац, јун 2019.

Ментор:
Др Ненад Филиповић, ред. проф.

_____

Садржај

Abstract

Development process of Billiard game using OpenGL library is described in this thesis. User interface development process is described in detail, with explanations of all used primitives and functions of OpenGL library. Also, Billiard rules and their implementation in application is explained, as well as mechanical laws that need to be applied to resolve the collision.

Key words: 8 ball pool, Collision, OpenGL, Computer graphics

Резиме

У овом раду је описан поступак развоја игрице Билијар употребом OpenGL библиотеке. Детаљно је описан поступак реализације корисничког интерфејса, уз појашњења начина коришћења примитива и функција OpenGL окружења. Такође су описана правила игре билијар и њихова имплементација у апликацији, као и механички закони које је потребно применити за решавање судара.

Кључне речи: Осмица, Судар, OpenGL, Рачунарска графика

1. Увод

У уводном делу биће дефинисана сврха и примена рачунарске графике, као и OpenGL-а - развојног окружења које је коришћено за израду овог рада. Такође ће бити дат и кратак опис и преглед игре билијар.

1.1. Рачунарска графика

Рачунарска графика подразумева креирање и обрађивање слика у две или три димензије, уз помоћ рачунара и софтвера који су за то предвиђени. Рачунарска графика служи за процесирање визуалних сигнала, детекцију ивица и издвајање линија, обраду текстура, представљање карактеристике сцене, покрет, стереовизију и разне методе за обраду слика. Примена рачунарске графике се проналази и у науци, индустрији, дизајну и архитектури. У рачунарску графику спада и дизајн интерфејса оперативних система и софтвера (посебно видео игара), као и интернет страница.

Основне поддисциплине рачунарске графике су:

- Моделовање – бави се описивањем раванских фигура и тела (на пример мрежним моделима)
- Рендеровање – процес креирања дигиталне слике на основу (дводимензионалног или тродимензионалног) модела и (реалистичног или нереалистичног) модела понашања светлости.
- Анимације – секвенцијално приказивање низа слика великом брзином које изгледа као глатко кретање.
- Обрада слика (енгл. *image processing*) – бави се реконструкцијом дводимензионалних или тродимензионалних објеката на основу њихових слика. Обрада слика укључује примене у сателитским снимањима, медицини, космичким истраживањима, роботици, препознавању слова и друго [1].

1.2. OpenGL

OpenGL (скраћеница појма Open Graphics Library) је вишеплатформска библиотека са подршком за велики број програмских језика, а чији је циљ омогућавање писања апликација које раде са 2Д и 3Д графиком. OpenGL библиотека дефинише низ примитива које се користе за изградњу сложених сцена (као што су тачка, линија и полигон) [2].

Такође, OpenGL нуди могућност примене различитих трансформација над објектима у сцени, нуди различите врсте пројекција, нуди одбацивање делова објеката који су посматрачу невидљиви, примену текстура и још низ различитих могућности.

Као додатак OpenGL-у, а са циљем пружања више могућности, често се уз њега користе и две помоћне библиотеке: GLU и GLUT.

Библиотека GLU (скраћеница појма OpenGL Utility Library) обогаћује скуп наредби које пружа OpenGL и уводи комплексније примитиве које је могуће користити у опису сцена, нпр. NURBS криве и површине, сфере и цилиндре.

Библиотека GLUT (скраћеница појма OpenGL Utility Toolkit) додатно поједностављује израду апликација које користе OpenGL увођењем платформски независне подршке за стварање и рад са прозорима, за хватање и обраду низа догађаја (догађаји миша или тастатуре), за израду менија и слично. Ова библиотека пружа и дефиниције још неких сложених објеката који су кориснику стављени на располагање, на пример торус и чајник [2].

1.3. Билијар

Билијар (осмица) је игра која се игра на равном столу правоугаоног облика на коме се налази шест рупа (на све четири ивице и по једна на средини дужих страна), а уз помоћ малих кугли и штапа. Укупан број кугли је 16 од којих је једна бела, а остале су различитих боја, могу бити пуне или шарене и означене су бројевима.

Играч користи билијарски штап којим удара белу куглу, а која потом погађа циљану куглу. Циљ игре јесте убацити све кугле из своје групе (или кугле од броја један до броја седам, или кугле од броја девет до броја петнаест) у једну од шест рупа које се налазе на столу пре убацивања кугле број осам.

Приказ билијарског стола на почетку игре приказан је на слици 1 [3].

Слика 1. Приказ билијарског стола

2. Историјат билијара

Билијар је друштвена игра која развија социјализацију, такмичарски дух, али и моторику, концентрацију и рефлексе. Могу да га играју људи свих година, а правила се лако уче.

Билијар је израз који се односи на широк распон игара вештине у којима се за игру користи билијарски штап, кугле и билијарски сто пресвучен вуненим платном (чојом), а уоквирен мартинелама (гуменим ивицама).

Игре које се сматрају билијаром деле се у три групе:

1. Карамбол – група игара које се играју на столовима без рупа
2. Билијар са рупама – углавном се игра на столовима са шест рупа. Подваријанте овог билијара су:
    1. **Осмица** (најпопуларнија билијарска игра на свету) – игра која је тема овог рада (8 ball pool)
    2. Деветка (9 ball pool)
    3. Десетка (10 ball pool)
3. Снукер – сврстава се у групу билијара са рупама, али се историјски разликује од осталих игара

Историја билијара је дугачка и веома интересантна прича. Откриће билијара се везује још за 11. век и крсташке ратове. У то време је билијар била игра која се играла уз помоћ штапова и лоптица, али на земљи. Била је слична крикету на основу којег се и развио билијар. Тек у 16. веку је билијар почео да се игра на једноставно ограђеним дрвеним столовима са зеленом подлогом. Сврха мартинела на крајевима стола јесте да се спречи испадање кугле ван стола. Играло се са две кугле, а постојали су и такозвани „прстенови“ или „обручи“ кроз које је требало „протерати“ кугле. У 18. веку престаје употреба обруча и прстенова, али остаје сто са рупама и куглама. Прве кугле су биле израђене од камена, док су се касније израђивале од дрвета [4].

Назив билијар потиче од француских израза *billart* што значи штап и *bille* што значи лопта [5].

У даљем тексту ће се под појмом „билијар” подразумевати игра осмица.

3. Правила билијара

У наредном одељку биће описана правила игре билијар која се односе на почетак игре, али и на могућа дешавања у току игре. Важна напомена је да у билијару не постоји бодовање потеза, већ се само прати стање убачених кугли [6].

Слагање кугли

На почетку игре, петнаест кугли се слаже унутар троугла тако да су све кугле што више прилепљене једна уз другу. Кугла на врху троугла се ставља на задњу тачку (енгл. *foot spot*), а кугла број осам у средину троугла. У последњем реду кугли, кугле које се налазе скроз лево и скроз десно морају бити из различитих група (једна пуна и једна шарена). Преостале кугле се ређају у троуглу без икаквог редоследа. Пример правилно поређаних кугли на почетку игре приказан је на слици 2 [7].

Слика 2. Пример правилно поређаних кугли на почетку игре

Почетни ударац

1. Бела кугла може да се постави било где иза основне линије
2. Бела кугла не мора да удари неку одређену куглу у троуглу
3. Сто је на почетку игре отворен
4. Ако играч убаци куглу из почетног ударца, сто престаје да буде отворен. Ако је ударац био правилан, играч наставља да игра

Отворен сто / Одабир групе кугли

Пре него што играчи одаберу своје групе кугли, за сто се каже да је „отворен“. Ако играч убаци куглу, кугле из те групе постају његове и његов противник добија другу групу кугли. Док је сто отворен, играч може прво да погоди било коју куглу осим кугле број осам, а да притом не направи прекршај.

Ток игре / Победа

1. Играч остаје за столом све док на легалан начин убацује кугле, односно не прави прекршај
2. Уколико је играч одиграо правилан потез, али ниједна кугла није убачена, на ред за игру долази његов противник
3. Играч побеђује ако убаци све кугле из своје групе, а након тога и куглу број осам без прављења прекршаја
4. Уколико играч направи прекршај, противник може белу куглу да стави на било које место на столу на ком се не налази друга кугла

Губитак партије

Играч губи партију када:

1. Направи прекршај приликом убацивања кугле број осам
2. Убаци куглу број осам пре него што је убацио све кугле из своје групе

Стандардни прекршаји

Ако играч направи прекршај, на ред за игру долази његов противник. Противник добија могућност да белу куглу стави на било коју позицију на столу и одатле игра. Прекршаји које играч може да направи у току игре су:

1. Бела кугла је прво ударила у куглу која припада противниковој групи (осим када је сто отворен)
2. Бела кугла је прво ударила у црну куглу
3. Бела кугла није ударила ни у једну куглу

4. Механика билијара

Механика која стоји иза билијара, великим делом се односи на судар две билијарске кугле. Када дође до судара између две кугле, тај судар се посматра као еластични судар. Због једноставности решавања проблема судара између две билијарске кугле, судар се често посматра као савршено еластичан.

4.1. Судар

Судар представља догађај у коме више тела делују једна на друге у кратком временском интервалу. При сударима су међусобне интеракције тела толико јаке да се све спољашње силе могу занемарити. Пошто спољашње силе које делују на тела током судара могу да се занемаре, такав систем се може сматрати изолованим.

Судар два тела је сложен процес који се тешко може пратити у свим његовим појединостима, па се зато приликом проучавања усвајају неке претпоставке које олакшавају анализу.

При анализи судара узима се у обзир да се масе пре и после судара не мењају и да су брзине пре судара познате. Брзине тела после судара могу се одредити применом закона одржања импулса и енергије. Закон одржања импулса може се применити у сваком анализираном случају, док се за примену закона одржања енергије разликују два случаја:

- Еластични судар – укупна кинетичка енергија тела при судару се не мења
- Нееластични судар – укупна кинетичка енергија тела мења се при судару [8]

Коефицијент реституције (коефицијент судара)

Коефицијент реституције k је мера реституције судара између два тела, односно овај коефицијент показује однос између кинетичке енергије која је задржана након судара и кинетичке енергије која је изгубљена, тј. претворена у други облик. Коефицијент реституције се дефинише као однос релативне брзине после и пре судара.

$$k = \frac{v'}{v} \tag{1}$$

Коефицијент реституције се креће у опсегу од 0 до 1.

Могуће вредности коефицијента реституције у зависности од судара:

- k = 0 – савршено нееластичан судар
- 0 < k < 1 – реалан нееластичан судар
- k = 1 – савршено еластичан судар [9]

4.2. Импулс и закон одржања импулса

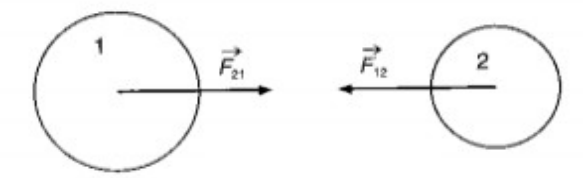
У физици се импулс означава малим словом $\vec{p}$. Импулс је једнак производу масе тела m и његове брзине $\vec{v}$, а јединица импулса је $kg\frac{m}{s}$.

$$\vec{p}=m\vec{v} \tag{2}$$

Како је брзина тела одређена њеним интензитетом, правцем и смером, а импулс зависи од брзине, онда се и импулс дефинише преко интензитета, правца и смера.

У општем случају, импулс неког тела концептуално се може посматрати као настојање тела да настави кретање у истом правцу и смеру, уколико на њега не делује нека спољашња сила.

Импулс је одржана величина, што значи да укупан импулс било ког изолованог система (који није под утицајем спољашњих сила) не може да се промени.
Посматра се, једноставности ради, изолован систем два тела у инерцијалном систему, која само узајамно делују један на други (слика 3).



Слика 3. Изолован систем два тела у инерцијалном систему

Маса првог тела је m_1 , брзина пре дејства $\vec{v}_1$, а након дејства $\vec{v}_1'$. Маса другог тела је m_2 , брзина пре дејства $\vec{v}_2$, а након дејства $\vec{v}_2'$. Одговарајући импулси тела су $\vec{p}_1$ и $\vec{p}_2$. Услед деловања сила, после временског интервала Δt импулси тела биће $\vec{p}_1'$ и $\vec{p}_2'$.

Према Другом Њутновом закону за узајамно дејство ових тела важи да је:

$$\vec{F}_{12}\Delta t=m_1\vec{v}_1'-m_1\vec{v}_1 \tag{3}$$

и

$$\vec{F}_{21}\Delta t=m_2\vec{v}_2'-m_2\vec{v}_2 \tag{4}$$

односно,

$$\frac{\vec{p}_1'-\vec{p}_1}{\Delta t}=\vec{F}_{21} \tag{5}$$

и

$$\frac{\vec{p}_2'-\vec{p}_2}{\Delta t}=\vec{F}_{12} \tag{6}$$

где је $\vec{F}_{12}$ сила којом прво тело делује на друго, а $\vec{F}_{21}$ сила којом друго тело делује на прво.

Како је према Трећем Њутновом закону $\vec{F}_{21}=-\vec{F}_{12}$, следи да је

$$\frac{\vec{p}_1{}'-\vec{p}_1}{\Delta t}=-\frac{\vec{p}_2{}'-\vec{p}_2}{\Delta t} \tag{7}$$

односно

$$\vec{p}_1{}'-\vec{p}_1=-\vec{p}_2{}'+\vec{p}_2 \tag{8}$$

и коначно

$$\vec{p}_1{}'+\vec{p}_2{}'=\vec{p}_1+\vec{p}_2 \tag{9}$$

На основу претходних једначина се може извести закон одржања количине кретања:

$$m_1\vec{v}_1+m_2\vec{v}_2+m_3\vec{v}_3+...+m_n\vec{v}_n=const \tag{10}$$

Укупан импулс изолованог система тела у инерцијалном референтном систему током времена се не мења:

$$\vec{p}_1+\vec{p}_2+...+\vec{p}_n=const \tag{11}$$

4.3. Кинетичка енергија и закон одржања енергије

Енергија је способност тела да врши рад, а њена јединица јесте џул (J).

Механичку енергију имају тела која се крећу, која се налазе у гравитационом пољу или су еластично деформисана.

Укупна механичка енергија тела једнака је збиру његове кинетичке и потенцијалне енергије:

$$E=E_k+E_p \tag{12}$$

Унутар изолованог система тела интерагују међусобно и крећу се – имају потенцијалну и кинетичку енергију.

Енергија се не може створити нити уништити, већ само прелази са једног тела на друго или се претвара из једног облика у други:

$$E=E_k+E_p=const \tag{13}$$

Ако су све унутрашње силе конзервативне, механичка енергија система се одржава. Кинетичке и потенцијалне енергије појединих тела у систему могу да се мењају, али збир кинетичких и потенцијалних енергија свих тела остаје исти у сваком тренутку.

Енергија коју тела имају при кретању је кинетичка енергија (E_k).

$$E_k=\frac{mv^2}{2} \tag{14}$$

Кинетичка енергија је сразмерна маси тела и квадрату његове брзине. Она је скаларна величина и не зависи од правца и смера кретања тела. Њена вредност не може бити негативна, а једнака је нули у стању мировања [10].

4.4. Примена закона одржања енергије и импулса на судар

Импулс има специјално својство да се у изолованим системима одржава чак и приликом судара тела у систему. Пошто се импулс одржава, његов закон одржања се обично користи да би се израчунале брзине тела након судара. Како се импулс увек одржава, сума импулса честица пре судара мора да буде једнака суми импулса после судара:

$$m_1 \vec{v}_{1,i} + m_2 \vec{v}_{2,i} = m_1 \vec{v}_{1,f} + m_2 \vec{v}_{2,f} \tag{15}$$

где „i" означава иницијалне (почетне) брзине тела пре судара, а „f" означавава финалне (крајње) брзине тела после судара. Да би се одредио импулс пре или после судара потрено је познавање врсте судара који се одиграо.

4.5. Еластични судари

Судар између две билијарске кугле је добар пример за скоро потпуно еластични судар. Осим што је импулс у овом судару одржан, и збир кинетичких енергија кугли пре судара мора бити једнак збиру кинетичких енергија после судара. Једначине које одређују кретање кугли пре и после судара су:

$$m_1 \vec{v}_1 + m_2 \vec{v}_2 = m_1 \vec{v}_1' + m_2 \vec{v}_2' \quad \text{закон одржања импулса} \tag{16}$$

$$\frac{1}{2} m_1 v_1^2 + \frac{1}{2} m_2 v_2^2 = \frac{1}{2} m_1 v_1'^2 + \frac{1}{2} m_2 v_2'^2 \quad \text{закон одржања енергије} \tag{17}$$

Тела могу при судару међусобно да размене енергију, али укупна механичка енергија система остаје иста.

Коначне брзине након једнодимензионалног судара налазе на основу формула изведених из горенаведених закона:

$$v_1' = \frac{m_1 - m_2}{m_1 + m_2} v_1 + \frac{2m_2}{m_1 + m_2} v_2 \tag{18}$$

$$v_2' = \frac{2m_1}{m_1 + m_2} v_1 + \frac{m_2 - m_1}{m_1 + m_2} v_2 \tag{19}$$

где је $\vec{v}_1'$ брзина прве кугле након судара, а $\vec{v}_2'$ брзина друге кугле након судара [11].

У случајевима када се тела сударају у више од једне димензије, као што су удари под косим углом, брзине се разлажу на ортогоналне компоненте, где је једна компонента попречна на раван судара, а друга компонента је у равни судара. Компоненте брзина које су у равни судара остају непромењене, док се брзине попречне на раван судара израчунавају на исти начин као у једнодимензионалном случају. На пример, у дводимензионалним сударима, импулс може да се разложи на x и y компоненту. Тада се врше прорачуни за сваку компоненту посебно, и комбинују се резултати да би се добио укупан векторски резултат. Интензитет овог вектора је коначни импулс изолованог судара [11].

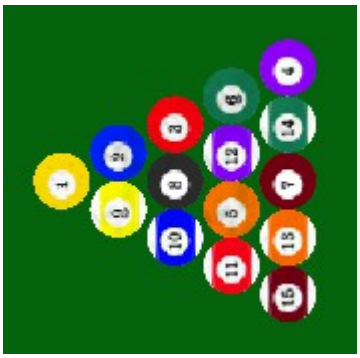
5. Имплементација игрице билијар у OpenGL графичком окружењу

У наредном одељку биће описан начин имплементације правила игре билијар која су описана у одељку 3. Такође ће бити приказан и изглед функција које су задужене за проверу исправности потеза играча.

5.1. Иницијално постављање кугли

Приликом израде апликације кугле су ређане на столу у складу са правилима описаним у одељку 3. Распоређивање кугли почиње тако што се црна кугла, односно кугла са бројем осам, смести на средину троугла. Након тога се насумично бира по једна кугла из групе пуних и једна из групе шарених и насумично стављају на крајњу леву или крајњу десну ивицу последњег реда троугла, а остале кугле се насумично распоређују на преостале позиције.

На слици 4 је приказан један од начина распоређивања кугли на почетку игре.



Слика 4. Распоред кугли на почетку игре

5.2. Позиционирање беле кугле

Позицију беле кугле одређује играч приликом почетног ударца, као и након направљеног прекршаја свог противника, а у складу са правилима билијара датим у одељку 3. Уколико је у току почетни ударац (први ударац у току партије), бела кугла може бити смештена било где иза основне беле линије, у супротном може бити смештена на било ком месту на столу на коме се не налази нека друга кугла. Уколико прекршај није направљен и не наступа почетни ударац, бела кугла остаје на месту на ком се тренутно налази.

Позиција беле кугле се одређује уз помоћ миша. Кугла се премешта по столу заједно са показивачем миша. Уколико корисник притисне леви тастер миша, а позиција на којој се у том тренутку налази бела кугла је валидна, та позиција се постававља као позиција беле кугле, у супротном ће се кугла и даље премештати заједно са померањем показивача миша док се не смести на валидну позицију.

За омогућавање манипулисања позицијом кугле употребом миша, потребно је дефинисати функцију (у овом случају је то функција passiveMouseFunc()) и у њој дефинисати понашање програма у случају померања миша. Ову функцију је потребно проследити функцији glutPassiveMotionFunc() у главној функцији програма. Пример кода функције passiveMouseFunc() приказан је на слици 5.

```
void passiveMouseFunc(int x, int y) {
    float xPos = ((float)x) / ((float)(WindowWidth));
    float yPos = ((float)y) / ((float)(WindowHeight));
    xPos = WindowWidth * xPos;
    yPos = WindowHeight * yPos;
    yPos = WindowHeight - yPos;

    if (gameSetup.whiteBallIsPlaced && gameSetup.gameOver == false) {
        float whiteBallX = gameSetup.balls[0].getX() * WindowWidth / Xmax;
        float whiteBallY = gameSetup.balls[0].getY() * WindowHeight / Ymax;
        float opposite = yPos - whiteBallY;
        float adjacent = xPos - whiteBallX;
        float rotation = atan2(opposite, adjacent);
        rotation = (rotation * 180) / PI - 90;
        stickRotationAngle = rotation;
        gameSetup.stick->setRotationAngle(rotation);
    }
    else if(gameSetup.gameOver == false){
        if (gameSetup.startOfGame) {
            if (xPos * Xmax / WindowWidth <= 4.8f - gameSetup.balls[0].getR()) {
                gameSetup.balls[0].setX(xPos * Xmax / WindowWidth);
            }
            else {
                gameSetup.balls[0].setX(4.6f);
            }
            gameSetup.balls[0].setY(yPos * Ymax / WindowHeight);
        }
        else {
            gameSetup.balls[0].setX(xPos * Xmax / WindowWidth);
            gameSetup.balls[0].setY(yPos * Ymax / WindowHeight);
        }
    }
    glutPostRedisplay();
}
```

Слика 5. Пример кода функције за манипулисање позицијом кугле употребом миша

За омогућавање манипулисања позицијом кугле употребом тастера на мишу, потребно је дефинисати функцију (у овом случају је то функција mouseFunc()) и у њој дефинисати понашање програма у случају притиска тастера миша. Ову функцију је потребно проследити функцији glutMouseFunc() у главној функцији програма. Пример кода функције mouseFunc() приказан је на слици 6.

```
void mouseFunc(int button, int state, int x, int y) {
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        if (gameSetup.whiteBallIsPlaced && gameSetup.gameOver == false) {
            gameSetup.stick->setIsHitted(true);
            gameSetup.player->setHittedBall(true);
            glutPostRedisplay();
        }
        else {
            gameSetup.whiteBallIsPlaced = true;
            for (int i = 1; i < 16; i++) {
                if (Ball::collision(gameSetup.balls[0], gameSetup.balls[i]))
                {
                    gameSetup.whiteBallIsPlaced = false;
                    break;
                }
            }
        }
    }
}
```

Слика 6. Пример кода функције за манипулисање позицијом кугле употребом тастера миша

5.3. Подела кугли

Док је сто отворен, у свакој рунди је потребно пратити да ли је нека кугла убачена у рупу. Уколико јесте, потребно је знати која је то кугла, односно којој групи припада убачена кугла. На основу групе убачене кугле, играч који је ту куглу убацио добија групу кугли којој припада убачена кугла, а противник добија другу групу кугли. За праћене убачених кугли се користи низ inHolePerRound у који се кугле смештају по редоследу упадања у рупу. За добијање прве убачене кугле потребно је узети први елемент овог низа.

5.4. Прекршаји

До промене играча који је на реду да игра долази уколико играч који је на потезу није убацио ниједну куглу или је направио прекршај. Уколико играч није убацио ниједну куглу, сви елементи низа inHolePerRound поменутог у одељку 5.3 ће бити једнаки нули. У овом случају позиција беле кугле остаје непромењена. Уколико је играч направио неки од стандардних прекршаја, као што је додиривање прво црне кугле (притом играч није убацио све кугле из своје групе) или кугле противника, на ред за игру долази његов противник и он може да позиционира белу куглу било где на столу. За праћење првог контакта беле кугле са неком другом куглом се користи променљива firstBallWhiteBallCollidedWith у коју се смешта број кугле са којом је бела кугла прво остварила контакт у текућој рунди. Да ли је тај потез валидан или не, проверава се у функцији wrongBallFirstCollision() приказаној на слици 7.

```
bool wrongBallFirstCollision() {
    bool allBallsIn = true;
    if (player->getBalls() == playerBalls::solidBall) {
        for (int i = 0; i < 7; i++) {
            if (ballsInHole[i] == 0) {
                allBallsIn = false;
                break;
            }
        }
        if (firstBallWhiteBallCollidedWith > 8) {
            return true;
        }
        if (firstBallWhiteBallCollidedWith == 8 && allBallsIn != true) {
            return true;
        }
    }
    else if (player->getBalls() == playerBalls::stripedBall) {
        for (int i = 7; i < 14; i++) {
            if (ballsInHole[i] == 0) {
                allBallsIn = false;
                break;
            }
        }
        if (firstBallWhiteBallCollidedWith < 8 && firstBallWhiteBallCollidedWith > 0) {
            return true;
        }
        if (firstBallWhiteBallCollidedWith == 8 && allBallsIn != true) {
            return true;
        }
    }
    if (player->getBalls() == playerBalls::null) {
        if (firstBallWhiteBallCollidedWith == 8) {
            return true;
        }
    }
    return false;
}
```

Слика 7. Функција за проверавање валидности првог судара беле кугле

Уколико играч у току потеза убаци куглу која није из његове групе, играч је направио прекршај и на ред за игру долази његов противник који може да постави белу куглу на било које место на столу. Функција која проверава да ли је играч направио ову врсту прекршаја је функција wrongBallInHole() приказана на слици 8.

```
bool wrongBallInHole() {
    if (player->getBalls() == playerBalls::solidBall) {
        for (int i = 0; i < 15; i++) {
            if (inHolePerRound[i] > 8) {
                return true;
            }
        }
    }
    else if (player->getBalls() == playerBalls::stripedBall) {
        for (int i = 0; i < 15; i++) {
            if (inHolePerRound[i] < 8 && inHolePerRound[i] > 0) {
                return true;
            }
        }
    }
    return whiteBallInHole;
}
```

Слика 8. Функција за проверавање убацивања противникове кугле

Уколико играч убаци црну куглу, потребно је проверити да ли је пре тога убацио све кугле из своје групе. Ако су све кугле из играчеве групе убачене, овај потез се не сматра прекршајем, већ доводи играча до победе уколико у току потеза не направи неку другу врсту прекршаја (на пример убаци и белу куглу). У супротном, играч је направио прекршај и губи партију. Функција која проверава да ли је играч направио ову врсту прекршаја је функција blackBallInHoleValidMove() приказана на слици 9.

```
bool blackBallInHoleValidMove() {
    bool validTurn = true;

    if (player->getBalls() == playerBalls::stripedBall) {
        for (int i = 0; i < 7; i++) {
            if (ballsInHole[i] != 1) {
                validTurn = false;
            }
        }
    }
    else if (player->getBalls() == playerBalls::solidBall) {
        for (int i = 7; i < 14; i++) {
            if (ballsInHole[i] != 1) {
                validTurn = false;
            }
        }
    }
    if (whiteBallInHole == true) {
        validTurn = false;
    }

    return validTurn;
}
```

Слика 9. Функција за проверавање валидности убацивања црне кугле

5.5. Остала правила

Уколико играч не убаци ниједну куглу у току потеза, на ред за игру долази његов противник, али бела кугла остаје на месту на коме се тренутно налази (уколико играч није направио неки други прекршај који захтева померање беле кугле). Уколико бела кугла не додирне ниједну куглу у току потеза, играч је направио прекршај и на ред за игру долази његов противник који може белу куглу да постави на било које месту на столу на коме се не налази нека друга кугла. Описани случајеви се проверавају после сваког потеза играча у функцији newRound() у оквиру које се позивају и изнад наведене функције за испитивање валидности потеза играча.

6. Опис корисничког интерфејса

Кориснички интерфејс апликације састоји се од једног прозора на коме је приказан билијарски сто на коме се налазе кугле, три штапа и преглед статуса игре, односно статуса убачених кугли. Такође је употребом слика играча који је на потезу, односно играча који чека да други играч одигра свој потез, омогућен увид у то који играч је тренутно на потезу (слика 10).

Слика 10. Кориснички интерфејс апликације

6.1. Билијарски сто

Билијарски сто је представљен као четвороугао са облим ивицама размера 2:1. Облик стола је нацртан на начин приказан на слици 11.

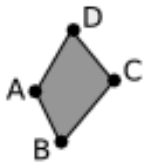
Слика 11. Цртање полигона са облим ивицама

Прво је потребно нацртати два правоугаоника чија ће ширина, односно дужина бити мања од жељене ширине, односно дужине стола за дужину дела који треба да буде кружног облика k, а који ће међусобно бити у односу као на слици. Након цртања два правоугаоника, на сваког тачки њиховог пресека потребно је нацртати по један круг полупречника k.

За реализацију оваквог начина цртања полигона употребљен је блок наредби који започиње наредбом glBegin(GL_QUADS) и завршава се наредбом glEnd(). Између та два граничника задате су по четири тачке које представљају ивице четвороуглова. Примитива GL_QUADS исцртава један четвороугао који је одређен задатим тачкама.

Начин рада примитиве GL_QUADS приказан је на слици 12.

```
glBegin(GL_QUADS);
glVertex2f(Ax, Ay);
glVertex2f(Bx, By);
glVertex2f(Cx, Cy);
glVertex2f(Dx, Dy);
glEnd();
```



Слика 12. Приказ рада примитиве GL_QUADS

За исцртавање облих ивица креиран је по један круг на све четири ивице са центром у тачки пресека два четвороугла нацртана на претходно описан начин. За спољну ивицу стола, односно за цртање наведених елемената, коришћена је браон боја.

Функција која је коришћена за постављање боје је функција glColor3f(float red, float green, float blue) која као аргументе прима три реална броја између 0.0 и 1.0. Ови аргументи дају црвену, зелену и плаву компоненту боји. 1 подразумева максимално пристуство боје, а 0 подразумева одсуство те боје. Омогућена је опција мешања све три боје ради добијања комплетног опсега боја. На пример, чисто црвена боја је представљена као (1, 0, 0), а чисто плава као (0, 0, 1). Бела боја је комбинација свих боја, па се она представља као (1, 1, 1), док је црна - одсуство свих боја, представљена као (0, 0, 0). Жута боја је комбинација црвене и зелене и представља се као (1, 1, 0). Наранџаста је жута са мање зелене, па се представља као (1, 0.5, 0). На исти начин се може добити било која боја. Пример кода за одабир боје примитиве приказан је на слици 13 [12].

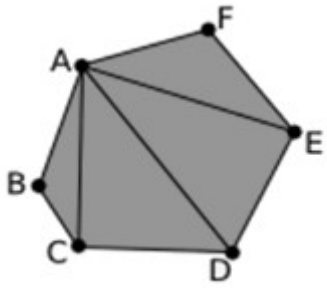
```
glColor3f(red, green, blue)
glColor3f(0.0,  0.0,  0.0);     црна
glColor3f(1.0,  0.0,  0.0);     црвена
glColor3f(0.0,  1.0,  0.0);     зелена
glColor3f(1.0,  1.0,  0.0);     жута
glColor3f(0.0,  0.0,  1.0);     плава
glColor3f(1.0,  0.0,  1.0);     љубичаста
glColor3f(0.0,  1.0,  1.0);     тиркизна
glColor3f(1.0,  1.0,  1.0);     бела
-------------------------------------------------------------
glColor3f(0.5,  0.0,  0.0);     тамно црвена
glColor3f(0.0,  0.5,  0.0);     тамно зелена
glColor3f(0.0,  0.0,  0.5);     тамно плава
```

Слика 13. Пример кода за одабир боје

Исцртавање унутрашњости стола и мартинела такође је реализовано употребом GL_QUADS примитиве и коришћењем зелене боје.

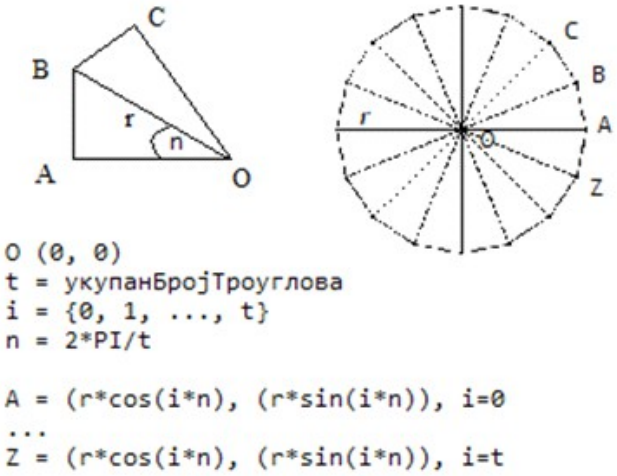
На столу се налази и шест рупа које су креиране цртањем кругова црне боје.

За реализацију цртања круга употребљен је блок наредби који започиње наредбом glBegin(GL_TRIANGLE_FAN) и завршава се наредбом glEnd(). Примитива GL_TRIANGLE_FAN од прве три унете координате прави троугао, а затим за сваку следећу координату додаје нови троугао, али у овом случају нови троугао је направљен повезивањем нове координате са претходном координатом и са првом унетом координатом (координата „A“ на слици 14).



Слика 14. Приказ рада примитиве GL_TRIANGLE_FAN

Испуњени круг се употребом примитиве GL_TRIANGLE_FAN може нацртати на начин приказан на слици 15.

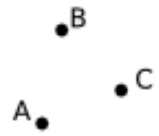


Слика 15. Математика потребна за цртање круга уз помоћ троуглова

За цртање круга уз помоћ примитиве GL_TRIANGLE_FAN потребно је нацртати довољан број троуглова како би круг био што правилнијег облика. Први корак јесте одређивање координате центра, у овом случају је центар постављен на координатни почетак (0, 0). Координате темена троуглова се одређују по формули приказаној на слици 15.

За цртање празног полукруга употребљен је блок наредби који започиње наредбом glBegin(GL_POINTS) и завршава се наредбом glEnd(). Примитива GL_POINTS исцртава тачке на екрану. Свака координата представља једну тачку. Начин рада примитиве GL_POINTS приказан је на слици 16.

```
glBegin(GL_POINTS);
glVertex2f(Ax, Ay);
glVertex2f(Bx, By);
glVertex2f(Cx, Cy);
glEnd();
```



Слика 16. Приказ рада примитиве GL_POINTS

Код за цртање полукруга коришћењем ове примитиве је дат на слици 17.

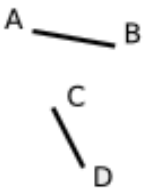
```
glTranslatef(4.8f, 4.5f, 0.0f);
glBegin(GL_POINTS);
for (int i = 200; i < 600; i++) {
    float angle = 3 * PI / 2 * i / 600;
    glVertex2f(1.5f * cos(angle), 1.5f * sin(angle));
}
glEnd();
```

Слика 17. Пример цртања полукруга коришћењем примитиве GL_POINTS

Бела линија на столу, односно основна линија, нацртана је употребом блока наредби који започиње наредбом glBegin(GL_LINES) и завршава се наредбом glEnd(). Примитива GL_LINES исцртава линије на екрану. Сваке две узастопне координате унутар овог блока наредби представљају једну линију.

Начин рада примитиве GL_ LINES приказан је на слици 18.

```
glBegin(GL_LINES);
glVertex2f(Ax, Ay);
glVertex2f(Bx, By);
glVertex2f(Cx, Cy);
glVertex2f(Dx, Dy);
glEnd();
```

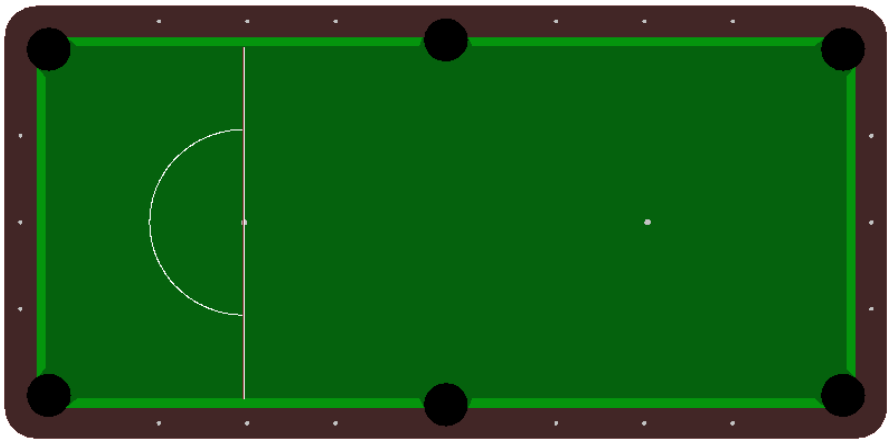


Слика 18. Приказ рада примитиве GL_ LINES

Дебљину исцртане линије могуће је променити употребом функције glLineWidth(float width) којој се као аргумент прослеђује жељена дебљина линије. Уколико се експлицитно не унесе вредност ове функције, њена вредност је постављена на 1.

Као и за све остале примитиве, и за примитиву GL_ LINES је могуће одабрати боју којом ће линија бити исцртана позивањем функције glColor3f(gloat red, float green, float blue) и прослеђивањем жељених параметара. У овом случају линија је беле боје, па је потребно проследити параметре (1, 1, 1).

Приказ билијарског стола апликације дат је на слици 19.



Слика 19. Приказ билијарског стола креиране апликације

6.2. Билијарске кугле

Билијарских кугли укупно има 16 и свака осим беле је означена бројем. Кугле могу бити пуне или шарене, што значи да су обојене целе или да су беле са пругом у одређеној боји. Бројеви и боје кугли у билијару су дати у табели 1.

Табела 1. Преглед боја и бројева билијарских кугли

Боја кугле	Број кугле
Бела	/
Жута	1
Плава	2
Црвена	3
Љубичаста	4
Наранџаста	5
Зелена	6
Браон	7
Црна	8
Жуто-бела	9
Плаво-бела	10
Црвено-бела	11
Љубичасто-бела	12
Наранџасто-бела	13
Зелено-бела	14
Браон-бела	15

Кугле означене бројевима од 1 до 7 су пуне кугле, а кугле означене бројевима од 9 до 15 су шарене. Бројем 8 је означена црна кугла која се убацује тек након што играч убаци све кугле из своје групе. Тежина ових кугли је око 170 грама, а пречник је око 57,15 mm [13].

Исцртавање кугли у апликацији је реализовано употребом gluSphere() функције која исцртава сферу задатог полупречника. За подешавање изгледа кугли коришћено је придруживање текстура сфери. Да би коришћење текстура било омогућено, најпре је потребно дозволити употребу 2Д текстура позивом функције glEnable(GL_TEXTURE_2D). Након омогућавања коришћења текстура потребно је креирати објекат типа QLUquadric и проследити га функцији gluQuadricDrawStyle као први аргумент, а као други аргумент ове функције треба проследити GLU_FILL као жељени стил цртања. Након тога је потребно одабрати 2Д текстуру која ће бити приказана на нацртаном објекту позивом функције glBindTexture(GL_TEXTURE_2D, текстура). Следећи корак јесте прослеђивање објекта типа QLUquadric функцијама gluQuadricTexture() и gluQuadricNormals() као први аргумент. Сада је потребно дефинисати саму сферу, као и њен полупречник. Овај корак подразумева позивање функције gluSphere() којој се као први аргумент прослеђује објекат типа QLUquadric, као други аргумент полупречник сфере, а као трећи и четврти аргумент се прослеђује број потподељака око и дуж z осе. Код за креирање сфере и придруживање текстуре сфери је дат на слици 20.

```
glEnable(GL_TEXTURE_2D);
glColor3f(1, 1, 1);
GLUquadric *sphere = gluNewQuadric();

gluQuadricDrawStyle(sphere, GLU_FILL);
glBindTexture(GL_TEXTURE_2D, texture);
gluQuadricTexture(sphere, GL_TRUE);
gluQuadricNormals(sphere, GLU_SMOOTH);
gluSphere(sphere, ball.r, 32, 16);
```

Слика 20. Код за креирање сфере и придруживање текстуре

Текстуре које су коришћене у апликацији су слике екстензије *.bmp*. Оне се учитавају у низ након покретања програма и прослеђују се као аргументи функцијама за исцртавање сфера [14].

Ротација кугли је имплементирана употребом функције glRotatef(float angle, float x, float y, float z). Као аргументи овој функцији се прослеђују угао ротације (у степенима), као и x, y и z координате вектора. Ова функција омогућава ротацију објекта за одређени угао око x, y или z вектора. У зависности од кретања кугле, рачуна се угао ротације који се прослеђује овој функцији приликом исцртавања кугле заједно за вектором ротације.

6.3. Билијарски штап

Билијарски штап се користи за ударање беле кугле. Постоји много различитих боја и дизајна штапа, док у конкретној апликацији постоји три штапа истог дизајна, али различитих боја (бели, зелени и црвени). Ови штапови се разликују према јачини којом ударају белу куглу.

Према правилима билијара играчу је дозвољено да мења штапове током меча. У складу са тим правилом, са десне стране екрана апликације су приказани штапови које је могуће одабрати у току игре. На слици 21 је дат изглед штапова који су на располагању играчима у току игре.

Слика 21. Приказ штапова доступних у игри

За реализацију исцртавања штапа коришћен је блок наредби који започиње наредбом glBegin(GL_QUADS) и завршава се наредбом glEnd(). Начин употребе примитиве GL_QUADS која је употребљена за исцртавање штапова детаљно је описан у одељку 6.1.

За подешавање изгледа штапова коришћено је придруживање текстура нацртаним штаповима. Да би коришћење текстура било омогућено, најпре је потребно дозволити употребу 2Д текстура позивом функције glEnable(GL_TEXTURE_2D). Након омогућавања коришћења текстура потребно је одабрати 2Д текстуру која ће бити приказана на нацртаном објекту позивом функције glBindTexture(GL_TEXTURE_2D, текстура). Следећи корак јесте позивање блока наредби за ицртавање штапа који је претходно описан. Разлика у односу на стандардно исцртавање правоугаоника јесте да се између наредби glBegin(GL_QUADS) и glEnd() уз прослеђивање координата правоугаоника прослеђују и координате текстуре позивањем функције glTexCoord2f(float s, float t), где су s и t координате текстуре у простору објекта за кога се текстура везује. Број 2 у називу ове функције означава да се ради о 2Д текстури. Код за креирање правоугаоника и придруживање текстуре истом је дат на слици 22.

```
static void draw_stick_left(Stick &stick, GLuint &texture, float x, float y) {
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();
    glTranslatef(x, y, 0);
    glColor3f(1, 1, 1);

    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texture);

    glBegin(GL_QUADS);
    glTexCoord2f(0.0, 1.0); glVertex3f(stick.x4, stick.y4, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(stick.x3, stick.y3, 0.0);
    glTexCoord2f(1.0, 0.0); glVertex3f(stick.x2, stick.y2, 0.0);
    glTexCoord2f(0.0, 0.0); glVertex3f(stick.x1, stick.y1, 0.0);
    glEnd();
    glDisable(GL_TEXTURE_2D);
}
```

Слика 22. Код за цртање штапа и придруживање текстуре

Текстуре које су коришћене у апликацији су слике екстензије *.bmp*. Оне се учитавају у низ након покретања програма и прослеђују се функцијама за исцртавање штапова.

На почетку игре, играчу је додељен бели штап, који најслабије удара куглу. Поред белог штапа постоје и зелени и црвени штапови. Избор штапа којим ће ударити куглу играч извршава на тастере 1, 2 и 3. Број 1 додељује играчу бели штап, број 2 зелени штап, а број 3 црвени штап.

Да би се омогућило манипулисање штаповима употребом тастера, потребно је дефинисати функцију (у овом случају је то функција keyboardFunc()) и у њој дефинисати понашање програма у зависности од притиснутог тастера. Када корисник притисне тастер, сваки тастер се преводи у ASCII карактер и позива се keyboardFunc() функција. Ову функцију је потребно проследити функцији glutKeyboardFunc() у главној функцији програма. Код за keyboardFunc() функцију је дат на слици 23.

```
void keyboardFunc(unsigned char key, int x, int y)
{
    switch (key) {
    case ' ':
        if (gameSetup.whiteBallIsPlaced && gameSetup.gameOver == false) {
            gameSetup.stick->setIsHitted(true);
            gameSetup.player->setHittedBall(true);
            glutPostRedisplay();
        }
        break;
    case 49: //number 1
        gameSetup.stick = &gameSetup.stick_first;
        gameSetup.stick->setRotationAngle(stickRotationAngle);
        glutPostRedisplay();
        break;
    case 50: //number 2
        gameSetup.stick = &gameSetup.stick_second;
        gameSetup.stick->setRotationAngle(stickRotationAngle);
        glutPostRedisplay();
        break;
    case 51: //number 3
        gameSetup.stick = &gameSetup.stick_third;
        gameSetup.stick->setRotationAngle(stickRotationAngle);
        glutPostRedisplay();
        break;
    case 27:
        exit(1);
    }
}
```

Слика 23. Пример кода функције за манипулисање штаповима употребом тастера

ASCII карактер 49 означава број 1 на тастатури, карактер 50 број 2, а карактер 51 број 3. ASCII карактер 27 означавава тастер esc (енгл. *escape*) на тастатури. За тастер за размак (енгл. *space*) уместо ASCII карактера искоришћен је знак празног места унутар апострофа.

На притисак тастера за размак омогућено је ударање беле кугле штапом. Други начин за ударање беле кугле јесте притиском левог тастера миша. Код за ударање кугле штапом на притисак левог тастера миша приказан је на слици 6 у одељку 5.2.

На притисак тастера 1, 2 или 3 врши се промена показивача stick у зависности од притиснутог тастера.

Ротација штапа је омогућена преко стрелица за лево и десно на тастатури, али и употребом миша. Први део кода са слике 5 (одељак 5.2) се односи на ротацију штапа у односу на положај показивача миша. У складу са померањем показивача миша, штап се ротира око беле кугле.

За омогућавање манипулисања стрелицама, потребно је дефинисати функцију (у овом случају је то функција specialKeyFunc()) и у њој дефинисати понашање програма у зависности од притиснутог тастера. Ову функцију је потребно проследити функцији glutSpecialFunc() у главној функцији програма. Пример кода функције glutSpecialFunc() приказан је на слици 24.

```
static void specialKeyFunc(int Key, int x, int y)
{
    switch (Key) {
    case GLUT_KEY_RIGHT:
        if (stickRotationAngle - stickStep > 0) {
            stickRotationAngle -= stickStep;
        }
        else {
            stickRotationAngle = 360 - (stickRotationAngle - stickStep);
        }
        gameSetup.stick->setRotationAngle(stickRotationAngle);
        break;
    case GLUT_KEY_LEFT:
        if (stickRotationAngle + stickStep < 360) {
            stickRotationAngle += stickStep;
        }
        else {
            stickRotationAngle = stickRotationAngle + stickStep - 360;
        }
        gameSetup.stick->setRotationAngle(stickRotationAngle);
    }
}
```

Слика 24. Пример кода функције за манипулисање штапом употребом стрелицама

6.4. Преглед статуса игре

Како у билијару не постоји бодовање потеза, у току игре је потребно омогућити играчима увид у број кугли из њихове групе које је потребно да убаце. Преглед преосталих кугли за убацивање обезбеђен је у сваком тренутку игре поред слика играча, након што се изврши подела кугли.

Увид у то који играч је на потезу обезбеђен је интуитивним сликама играча билијара изнад стола са леве и десне стране поред којих се налазе и кугле након расподеле између играча (слика 25).

Слика 25. Преглед убачених кугли и редоследа играња

6.5. Имплементација механике у апликацији

За имплементацију механике судара у апликацији, после сваког помераја кугле потребно је испитати да ли је дошло до судара са неком другом куглом. Испитивање се врши провером растојања између центара кугли. Уколико је растојање између центара две кугле мање или једнако збиру полупречника кугли, дошло је до судара. Функција која испитује да ли је дошло до судара је функција collision() (слика 26).

```
static bool collision(Ball &ball1, Ball &ball2) {
    float distance = sqrt(pow(ball1.xNew - ball2.xNew, 2) + pow(ball1.yNew - ball2.yNew, 2));

    if (distance <= ball1.r + ball2.r)
    {
        return true;
    }
    return false;
}
```

Слика 26. Функција за проверавање судара две кугле

Како је судар билијарских кугли еластичан, за рачунање нових брзина кугли се користе формуле за решавање еластичног судара. Фунција за решавање судара је фунција resolveCollision() (слика 27).

```
static void resolveCollision(Ball &ball1, Ball &ball2) {
    float dx = ball1.xNew - ball2.xNew;
    float dy = ball1.yNew - ball2.yNew;
    float nLenght = sqrt(dx * dx + dy * dy);
    float mtdx = dx * ((2 * ball1.r - nLenght) / nLenght);
    float mtdy = dy * ((2 * ball1.r - nLenght) / nLenght);
    ball1.xNew = ball1.xNew + mtdx * 0.5;
    ball1.yNew = ball1.yNew + mtdy * 0.5;
    ball2.xNew = ball2.xNew - mtdx * 0.5;
    ball2.yNew = ball2.yNew - mtdy * 0.5;

    dx = ball1.xNew - ball2.xNew;
    dy = ball1.yNew - ball2.yNew;

    float collisionAngle = atan2(dy, dx);
    float speed1 = sqrt(ball1.velocity_x * ball1.velocity_x + ball1.velocity_y * ball1.velocity_y);
    float speed2 = sqrt(ball2.velocity_x * ball2.velocity_x + ball2.velocity_y * ball2.velocity_y);

    float direction1 = atan2(ball1.velocity_y, ball1.velocity_x);
    float direction2 = atan2(ball2.velocity_y, ball2.velocity_x);
    float velocityx_1 = speed1 * cos(direction1 - collisionAngle);
    float velocityy_1 = speed1 * sin(direction1 - collisionAngle);
    float velocityx_2 = speed2 * cos(direction2 - collisionAngle);
    float velocityy_2 = speed2 * sin(direction2 - collisionAngle);
    float final_velocityx_1 = ((ball1.mass - ball2.mass) * velocityx_1 +
        (ball2.mass + ball2.mass) * velocityx_2) / (ball1.mass + ball2.mass);
    float final_velocityx_2 = ((ball1.mass + ball1.mass) * velocityx_1 +
        (ball2.mass - ball1.mass) * velocityx_2) / (ball1.mass + ball2.mass);
    float final_velocityy_1 = velocityy_1;
    float final_velocityy_2 = velocityy_2;

    ball1.velocity_x = (cos(collisionAngle) * final_velocityx_1 + cos(collisionAngle + PI / 2) * final_velocityy_1);
    ball1.velocity_y = (sin(collisionAngle) * final_velocityx_1 + sin(collisionAngle + PI / 2) * final_velocityy_1);
    ball2.velocity_x = (cos(collisionAngle) * final_velocityx_2 + cos(collisionAngle + PI / 2) * final_velocityy_2);
    ball2.velocity_y = (sin(collisionAngle) * final_velocityx_2 + sin(collisionAngle + PI / 2) * final_velocityy_2);

    ball1.velocity_x *= (1 - ball1.cof_ball_ball);
    ball1.velocity_y *= (1 - ball1.cof_ball_ball);
    ball2.velocity_x *= (1 - ball2.cof_ball_ball);
    ball2.velocity_y *= (1 - ball2.cof_ball_ball);

    ball1.xNew += ball1.velocity_x;
    ball2.yNew += ball1.velocity_y;
    ball2.xNew += ball2.velocity_x;
    ball2.yNew += ball2.velocity_y;
}
```

Слика 27. Функција за решавање судара између две кугле

Коефицијент реституције судара две лопте износи 0.06, коефицијент реституције лопте и штапа је 0.7, а коефицијент реституције лопте и стола је 0.5 [15].

Приликом решавања судара две кугле, прво је потребно израчунати угао судара. Након тога треба одредити правац и смер кретања кугли пре судара. Закони који се користе за одређивање нових брзина јесу закон одржања импулса и закон одржања енергије. Брзине је потребно разложити на тангенцијалну и нормалну компоненту. Тангенцијална компонента остаје непромењена, док се нормална компонента мења након судара. Нормална компонента након судара се рачуна коришћењем формула 18 и 19 из одељка 4.5.

Решавање удара кугле штапом и удара кугле у сто урађени су коришћењем истих закона као и за решавање судара две кугле. Функција која се користи за решавање удара кугле у сто је функција collisionWithTable. За одређивање брзине, правца и смера кугле након удара у сто потребно је разложити брзину пре удара на тангенцијалну и нормалну компоненту. Тангенцијална компонента остаје непромењена, док се нормална компонента израчунава употребом поменутих формула.

Функција која се позива након ударања кугле штапом је функција hitBall(). У овој функцији се одређује угао под којим је штап ударио белу куглу и у зависности до тога се рачуна брзина, правац и смер кретања беле кугле (слика 28).

```
static void hitBall(Ball &ball, Stick &stick) {
    float collisionAngleDegrees = stick.getRotationAngle();
    float collisionAngle = (collisionAngleDegrees * PI) / 180;

    ball.velocity_x = stick.getPower() * cos(collisionAngle);
    ball.velocity_y = stick.getPower() * sin(collisionAngle);

    ball.velocity_x *= 1 - ball.cof_stick_tip_ball;
    ball.velocity_y *= 1 - ball.cof_stick_tip_ball;

    ball.xNew = (ball.xNew + ball.velocity_x);
    ball.yNew = (ball.yNew + ball.velocity_y);
}
```

Слика 28. Функција за решавање удара кугле штапом

6.6. Коначан изглед апликације на крају игре

Игра траје све док један од играча не убаци црну куглу. На основу правила описаних у одељку 3 се одређује победник игре. Када наступи крај игре, на средини екрана се исписује ко је победио у одиграној партији.

За исписивање текста на екрану коришћена је функција drawText(). Овој функцији се прослеђује текст који треба бити исписан, број карактера у тексту, као и x и y координате на којима је потребно исписати текст. Поступак исписивања текста на екрану започиње позивом методе glMatrixMode(GL_PROJECTION) којој се прослеђује матрица пројекције.

Осим матрице пројекције могуће је одабрати и матрицу модела (GL_MODELVIEW). OpenGL користи појам одабране матрице и све матричне операције делују над том матрицом [16].

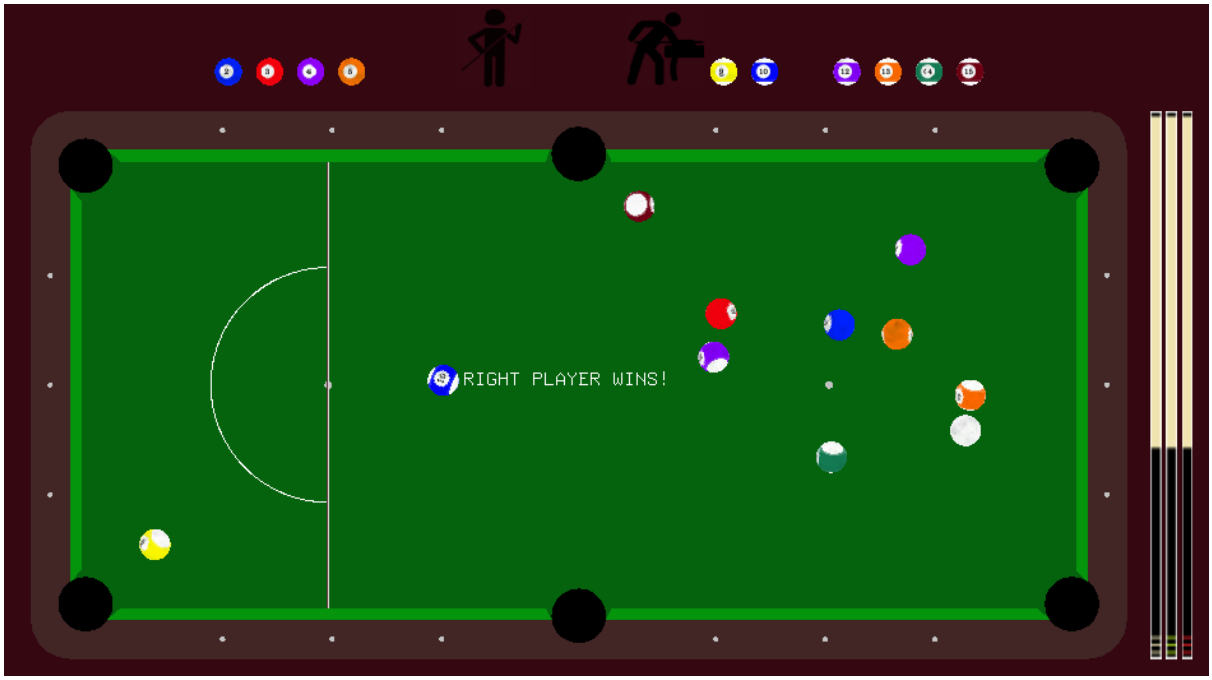
Након одабира матрице пројекције, потребно је позвати функцију glOrtho() којој се прослеђују координате текста. Након прослеђивања координата, методи glMatrixMode() се прослеђује матрица модела. Следећи корак јесте одређивање положаја растера (битмапе) за операције у пикселима. Након подешавања свих потребних параметара, може се приступити самом исписивању текста позивањем функције glutBitmapCharacter() која приказује битмап знак. Овој функцији се као аргументи прослеђују фонт којим ће карактер бити исписан и сам карактер. У конкретном случају је коришћен фонт GLUT_BITMAP_9_BY_15 – фонт фиксне ширине чији се сваки знак уклапа у правоугаоник величине 9x15 пиксела. Као и свакој примитиви поменутој до сада, и тексту се може одредити боја позивањем функције glColor3f(), што је у конкретном случају бела боја.

Исписивање текста се извршава када неко од играча убаци црну куглу у рупу, а након одређивања победника у функцији newRound(). Функција за исписивање текста на екрану је приказана на слици 29.

```
void drawText(const char* text, int length, int x, int y) {
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glLoadIdentity();
    glOrtho(0, 800, 0, 600, 0, 1);

    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();

    glRasterPos2i(x, y);

    for (int i = 0; i < length; i++) {
        glutBitmapCharacter(GLUT_BITMAP_9_BY_15, (int)text[i]);
    }

    glPopMatrix();
    glMatrixMode(GL_PROJECTION);
    glPopMatrix();
    glMatrixMode(GL_MODELVIEW);
}
```

Слика 29. Пример кода функције за исписивање текста на екрану

На слици 30 је приказан изглед апликације када наступи крај партије (у конкретном примеру је победник играч са десне стране).



Слика 30. Изглед апликације на крају партије

У ситуацији приказаној на слици 30 играч са леве стране је направио прекршај убацивања црне кугле пре убацивања свих кугли из своје групе, што је проузроковало крај партије и победу играча са десне стране.

7. Закључак

Као што се може закључити из претходног дела, за развој игрице попут билијара потребно је, поред познавања софтверских алата за рачунарску графику, познавати и механичке законе по којима се одвија игра како би апликација изгледала што реалније. Такође, неопходно је познавање и самих правила игре, као и потребних реквизита за исту, јер, као што је већ напоменуто, постоји више варијанти игрице билијар.

Унапређењу ове апликације може допринети пребацивање целе игре у 3Д, додавање осветљења, убацивање звука, промена перспективе и слично. Такође, побољшању би допринело и омогућавање кориснику да одабере коју варијанту билијара жели да игра и могућност играња турнира или играња путем интернета. У току даљег развоја ове игрице, може се додати опција играња против компјутера.

Унапређење са механичке стране би се могло реализовати омогућавањем кориснику да одабере неку другу позицију за удар беле кугле, а да то не буде њен центар како би могао да креира одређени спин кугли.

Поред забаве, ова апликација корисницима може послужити за учење правила билијара, као и за развијање такмичарског духа и стратешког начина размишљања. Када се играчи такмиче један против другог они покушавају да направе „ментални модел” намера противника, односно претпоставку како супротна страна размишља или који би могао бити следећи најлогичнији потез.

8. Литература

[1] Јаничић, П. (2014). *Рачунарска графика.* Београд: Математички факултет.

[2] Чупић, М.,&Михајловић, Ж. (2014). *Интерактивна рачунална графика кроз примјере у OpenGL-у.*

[3] *Неименована слика билијарског стола.* Преузето 16. августа 2019, са https://www.seanwoolsey.com/shop/woolsey-billiards-table

[4] *Историјат билијара*. Преузето 20. августа 2019, са http://skolabilijara.yolasite.com/istorija.php

[5] Миловановић, М. *Билијар - Популарна игра за све узрасте*. Преузето 01. септембра 2019, са https://betty.rs/bilijar-2/

[6] *Осмица.* Преузето 02. септембра 2019, са http://bilijar.rs/8-ball/

[7] *Један од многобројних начина правилног ређања кугли.* Преузето 05. септембра 2019, са https://en.wikipedia.org/wiki/Eight-ball

[8] *Еластични и нееластични судари.* Преузето 10. септембра 2019, са https://fizis.rs/гимназија/i-разред/закони-гравитације/elasticni-i-neelasticni-sudari/

[9] Врбић, И. *Експериментално утврђивање коефицијента реституције.* Преузето 15. септембра 2019, са https://repozitorij.vuka.hr/islandora/object/vuka%3A433/datastream/PDF/view

[10] *Закон одржања енергије у механици.* Преузето 18. септембра 2019, са https://fizis.rs/гимназија/i-разред/закони-гравитације/zakon-odrzanja-energije-u-mehanici/

[11] Којић, М. (1985). *Динамика – теорија и примери.* Београд: Научна књига

[12] *Увод у OpenGL.* Преузето 21. септембра 2019, са https://slideplayer.com/slide/8539015/

[13] Owoseni O. (2004). Тежина билијарске кугле. Преузето 25. септембра 2019, са https://hypertextbook.com/facts/2004/OluwoleOwoseni.shtml

[14] Zhongyn. *8 Ball Pool Game With OpenGL & C++.* Преузето 28. септембра 2019, са http://zhongyaonan.com/opengl-8-ball-pool-game/

[15] *Билијар.* Преузето 30. септембра 2019, са http://www.physics.usyd.edu.au/~cross/Billiards.htm

[16] *glMatrixMode.* Преузето 30. септембра 2019, са https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/glMatrixMode.xml