**Target Shooting**

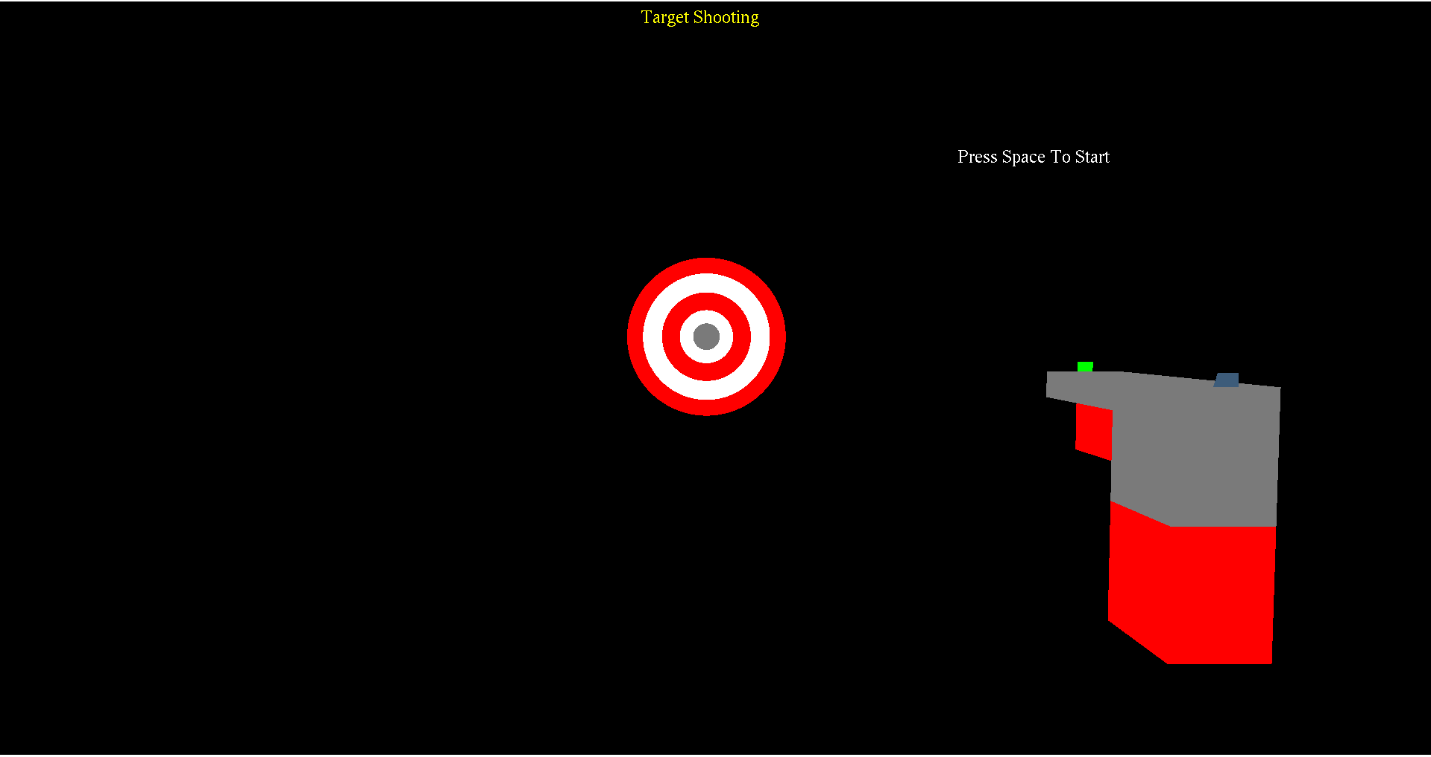
**שם המגיש: איתי סרוסי  
כיתה: יב' 1  
ת.ז: 324260009  
שם המנחה: דור כהן  
תאריך: 28.5.2019**

תוכן עניינים:

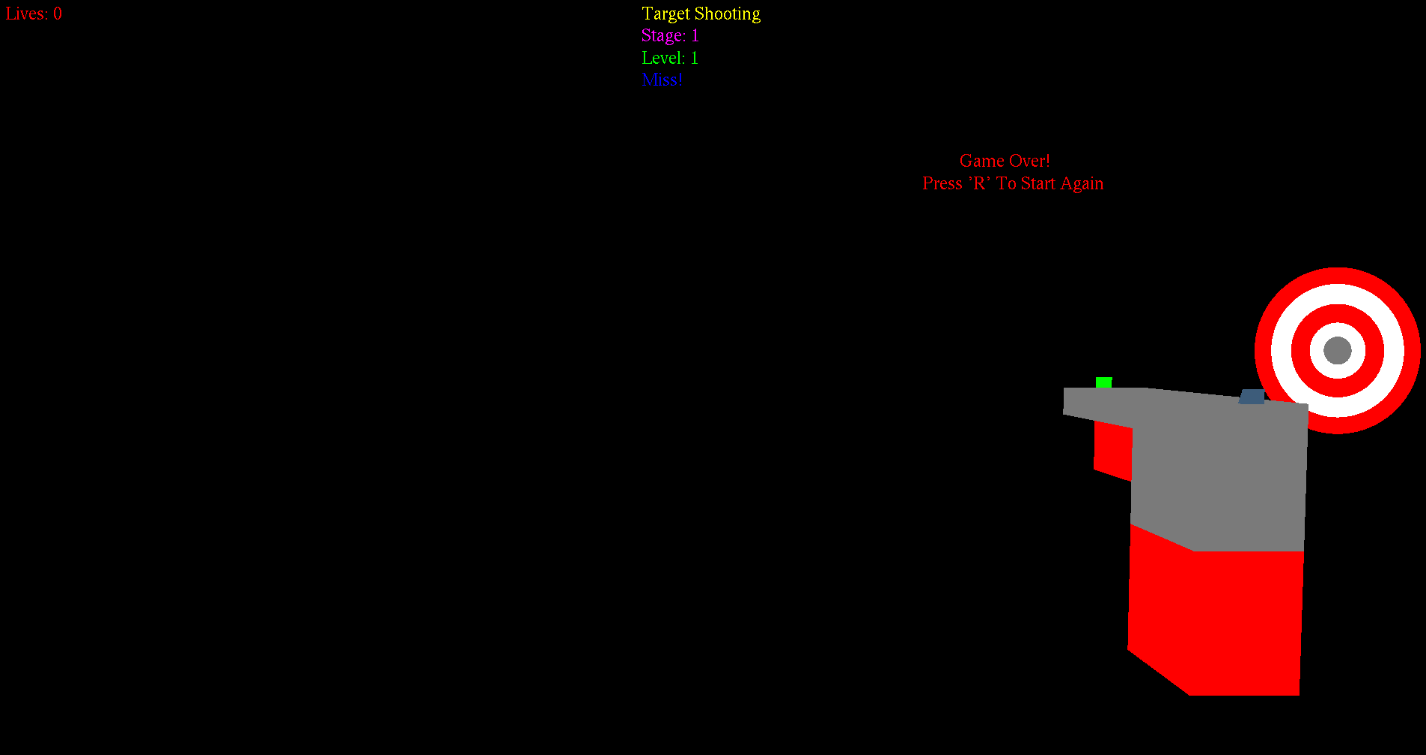
שער................................................................................1  
תוכן עניינים......................................................................2  
מדריך למשתמש...............................................................3  
מדריך למפתח..................................................................7  
פירוט מחלקות..................................................................8  
תרשים זרימה...................................................................16  
סיכום אישי.......................................................................17  
קוד התכנית..............................................................18 והלאה.

מדריך למשתמש:  
בכדי להריץ את הפרויקט יש להיכנס לתיקייה של הפרויקט  
(Itay's Project), להיכנס לאחר מכן לתיקייה Game, ולאחר מכן ללחוץ על הקובץ Target Shooting .



לאחר הלחיצה על הקובץ, ייפח הפרויקט, ויוצג החלון הבא:  
זהו המסך ההתחלתי של המשחק, **על מנת להתחיל לשחק**, כפי שרשום על המסך, יש ללחוץ על רווח(Space).  
  
  
  
  
  
  
  
לאחר שלחצת על רווח(Space), יתחיל המשחק ותעבור למסך הבא:  


זהו בעצם המסך המרכזי של המשחק, ופה בעצם יתבצע ה"שיחוק" או בעצם המשחק עצמו.  
המטרה תתחיל לזוז מקצה לקצה ומשחקיות עובדת כך:  
החצים ימינה ושמאלה: יזיזו את האקדח ימינה ושמאלה בהתאמה.  
מקש הרווח: ירה את הכדור מן האקדח.  
המקש R: יאתחל את המשחק למצב ההתחלתי שלו.(ראה תמונה קודמת).  
על המסך יוצגו הנתונים הבאים: במרכז המסך מתחת לשם המשחק (Target Shooting) יוצג מספר השלב(Stage) ומתחתיו מספר הרמה בשלב(Level).  
מצד ימין אל הנתונים הללו יוצגו הוראות המשחק.  
במידה והמשתמש לחץ על רווח(Space) ופגע במטרה, המשתמש יעלה רמה אחת בשלב ומהירות המטרה תגדל, במידה והחטיא ירד מספר החיים ב1.

  
בפינה השמאלית העליונה יוצגו מספר ה"חיים" שנותרו למשתמש באותו משחק(ערך התחלתי: 10), כאשר ערך החיים (Lives) יגיע ל-0 המשתמש ייפסל, המשחק ייגמר ויופיע המסך הבא:

כדי להתחיל מחדש את המשחק יש ללחוץ על הכפתור R.  
  
במידה והמשתמש לא נפסל וממשיך במשחק ישנם במשחק 3 שלבים(Stages):(לכל שלב 10 רמות)  
Stage 1: המטרה תזוז עד כאשר המשתמש יורה (לוחץ על רווח) המטרה תעצור במקומה, לאחר שהכדור נורה(בין אם פגע או לא) המטרה תמשיך לזוז.(המטרה תעצור רק כאשר הכדור נורה).  
Stage 2: המטרה תזוז תמיד (גם כשהמשתמש יורה).  
Stage 3: המטרה תזוז תמיד אך בנוסף – כאשר המשתמש יורה (לוחץ על רווח) כיוון תזוזת המטרה יתהפך.

אם במידה והמשתמש עבר את כל עשרת הרמות בכל שלושת השלבים המשתמש בעצם סיים את המשחק בהצלחה ויוצג המסך הבא:  
במסך זה בעצם תוצג הודעה של "מזל טוב"(Congratulations) על סיום המשחק, ועל מנת להתחיל מחדש את המשחק יש ללחוץ על הכפתור R.

מדריך למפתח:

**נתוני המערכת וסביבת העבודה:**

השפה בה מומש הפרויקט היא בשפת התכנות ++C.  
סביבת העבודה: Visual Studio 2017, בבית הספר ובבית.

הספריות בהן היה שימוש בפרויקט הן: OpenGL, glut.

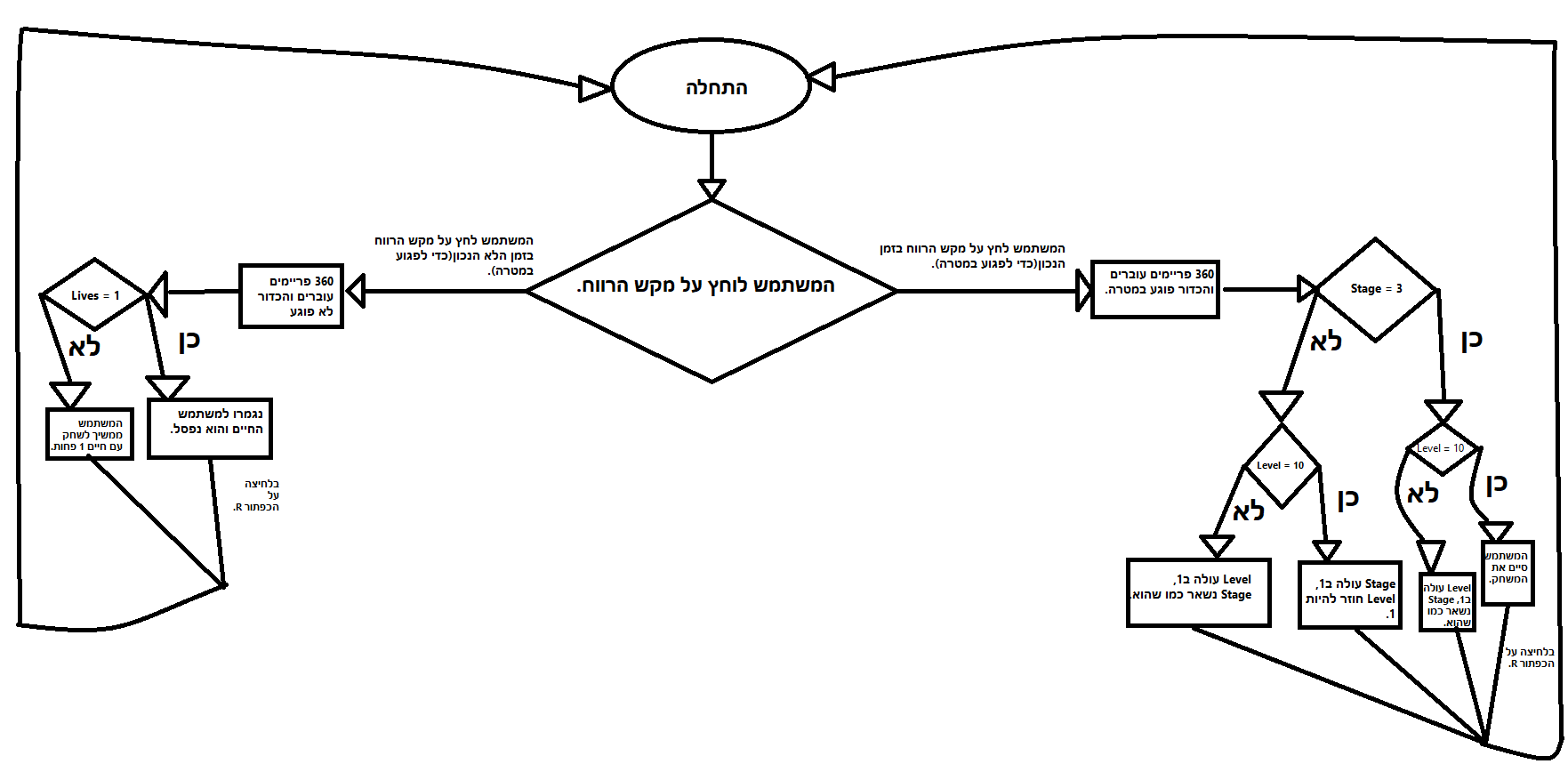
קבצים בשימוש:

|  |  |
| --- | --- |
| **קבצי header (.h)** | **קבצי source (.cpp)** |
| main.h | main.cpp |
| Target.h | Target.cpp |
| Gun.h | Gun.cpp |
| GameState.h | GameState.cpp |
| CPolygon.h | CPolygon.cpp |
| CModel.h | CModel.cpp |
| CPoint.h | CPoint.cpp |
| CMyOpenGLInit.h | CMyOpenGLInit.cpp |
| CCircle.h | CCircle.cpp |
| CBall.h | CBall.cpp |
| Bullet.h | Bullet.cpp |
| CGameHandler.h | CGameHandler.cpp |

|  |  |  |  |
| --- | --- | --- | --- |
| **שם הפונקציה:** | | **פרמטרים:(מה הפונקצייה מקבלת)** | **פירוט:**  **CGameHandler:** |
| StartOfGame() | |  | הפונקציה מאתחלת את כל הערכים שצריכים אתחול בתחילת המשחק. |
| DrawScence() | |  | הפונקציה מציירת את האובייקטים של המשחק על מסך המשחק. |
| LogicPart() | |  | אחראית לביצוע פעולות המשחק בכל פריים. |
| WriteText() | |  | אחראית לרשימת כל הטקסט על מסך המשחק. |
| KeyPressed() | | char key | הפונקציה אחראית לביצוע פעולות אשר המשתמש עושה בעת לחיצתו על הכפתורים, \*משתמשת בכפתורים המוגדרים על ידיי visual studio ללא ספריות מיוחדות. |
| SpecialKeyPressed() | | char key | הפונקציה אחראית לביצוע פעולות אשר המשתמש עושה בעת לחיצתו על הכפתורים, \*משתמשת בכפתורים מספריית glut. |
| GetBeckgroundColorR() | |  | הפונקציה מחזירה את ערך ה"red" של צבע הרקע |
| GetBeckgroundColorG() | |  | הפונקציה מחזירה את ערך ה"green" של צבע הרקע |
| GetBeckgroundColorB() | |  | הפונקציה מחזירה את ערך ה"blue" של צבע הרקע |
| ChangeBackgroundColors() | | float r, float g, float b | הפונקציה משנה את ערכי הצבעים לערכים שקיבלה |
| משתנים במחלקה: | | GameState gs1;  int BulletShotTimer = 360;  char Level[128];  char HP[128];  char stages[128];  Gun gun1;  Bullet bullet; Target target; | אחראי לבקרת המשחק. אחראי לטיימינג של יריית הכדור. אחראים לכתב שנראה על המסך(chars).  האקדח. הכדור. המטרה. |
|  | |  | **GameState:** |
| GameState() | |  | הפונקציה יוצרת הופעה חדשה של המחלקה. ומאתחלת את ערכי המשחק. |
| Gameover() | |  | הפונקציה בעצם מודיעה על סיום המשחק (לא בהצלחה) המשתמש בעצם נפסל ומגיע למסך "Gameover " שאותו הפונקציה הזאת מפעילה. |
| StageChange() | |  | הפונקציה אחראית על שינוי הסטייג'ים (השלב) |
| GameFinished() | |  | הפונקציה בעצם מודיעה על סיום המשחק (בהצלחה) המשתמש בעצם סיים את המשחק ומגיע למסך "Congratulations" שאותו הפונקציה הזאת מפעילה. |
| Stage2LivePlus1() | |  | הפונקציה מוסיפה למשתמש חיים 1 על הגיעו לשלב 2. |
| Stage3LivePlus3() | |  | הפונקציה מוסיפה למשתמש 3 חיים על הגיעו לשלב 3. |
| RESET() | |  | הפונקציה מאתחלת את כל ערכי המחלקה, ובעצם אנו משתמשים בה כדי לאתחל את המשחק על ידי לחיצה על הכפתור 'r' . |
| **\*בנוסף יש לכל המשתנים במחלקה פעולות - Get ו- Set .** | | המשתנים במחלקה: חיים. קבוע מהירות המטרה. קבוע מהירות תזוזות האקדח. משתנה המתריע על פסילה מוחלטת. מספר השלב. מספר הרמה. מכפיל מהירות המטרה. משתנה המתריע על סיום המשחק. משתנה המתריע על עליית שלב. משתנה המתריע על פספוס המטרה. משתנה המתריע על פגיעה במטרה. משתנה המתריע על פגיעה מדויקת. משתנה המתריע על התחלת המשחק. משתנה שאומר אם נורתה כבר ירייה. משתנה שאומר אם היה כבר ריסט. משתנה שאומר שהכדור כעת בהנפשת ירייה. | int lives;  double xChange;  double gunMove;  bool GameOver;  int Stage;  int Level\_Number;  double xChange\_Multiplier;  bool Congratz;  bool StageUp;  bool ShotMissed;  bool BulletHitTarget;  bool BullsEye;  bool GameStarted;  bool FirstShotBeenShot;  bool GameReboot; bool BULLETbeenSHOT; |
|  | |  | **Gun:** |
| למחלקה פונקציה אחת (פונקציה בונה) והיא: Gun() | |  | הפונקציה בעצם בונה את האקדח במשחק שבנוי מ31 פוליגונים ובנוי בתלת מימד מכל הכיוונים כך שלא משנה מאיפה יסתכלו על האובייקט הוא יראה תלת מימדי. |
|  | |  | **Bullet:** |
| Bullet(); | |  | המחלקה Bullet יורשת מהמחלקה CBall ובעצם הפעולה הבונה של Bullet מפנה אותה לפעולה הבונה של CBall ולכן Bullet הוא בעצם CBall. |
| BulletShot(); | |  | הפונקציה בעצם מבצעת את האנימציה/ הנפשה של יריית הכדור והזזתו במרחב המשחק. |
| BulletComeBack(); | |  | לאחר הירייה של הכדור, הכדור חוזר אל קנה האקדח, פעולה זו מבצעת את החזרה של הכדור לקנה. |
| BulletHitCheck(); | | GameState &gs1, Target &target | הפונקציה בעצם מקושרת לערכים הנמצאים בGameHandler ובעצם בודקת האם הכדור פוגע במטרה או לא, ואם הוא פוגע במרכזה, לפי פונקציה זו בעצם נכתבים טקסטים שונים על המסך כמו "BULLS EYE! " או "Miss!" ... |
|  | |  | **Target:** |
| Target(); | |  | המחלקה Target יורשת מהמחלקה CModel ולכן Target היא בעצם Model, הפונקציה הבונה של המחלקה בעצם בונה מודל שבנוי מ5 עיגולים שיוצרים את המטרה במשחק. |
| TargetMovement(); | | GameState &gs1 | פונקציה זו אחראית על תזוזת המטרה, בכל אחד מן השלבים והרמות השונות. |
|  |  | **CCircle:** |
| CCircle(); |  | הבנאי בעצם יוצר מעגל בסיסי ממספר נקודות שהוגדר התחלתית. |
| CCircle(); | int num\_of\_points | בנאי נוסף שבונה את העיגול לפי מספר הנקודות שקיבל. |
| SetNumberOfPoints(); | int num\_of\_points | משנה את מספר הנקודות שלפיהן נוצר העיגול. |
|  |  | **CPoint:** |
| CPoint(); |  | הבנאי הבסיסי של הפונקציה, יוצר הופעה חדשה ומאתחל את כל המשתנים של המחלקה. |
| CPoint(); | double x\_coordinate, double y\_coordinate, double z\_coordinate | בנאי נוסף, יוצר הופעה חדשה אך רק עם ערכי מיקום ובלי ערכי צבע. |
| CPoint(); | double x\_coordinate, double y\_coordinate, double z\_coordinate double red\_component double green\_component double blue\_component | בנאי נוסף, יוצר הופעה חדשה עם ערכי המיקום וגם עם ערכי הצבע. |
| MovePoint(); | CPoint point\_move | הפונקציה מזיזה את הנקודה לקורדינטות של הנקודה שקיבלה. |
| MovePoint(); | Axis axis, double move\_value | הפונקציה מזיזה את הנקודה בערך מסוים בציר המבוקש. |
| MovePoint(); | double x\_move, double y\_move, double z\_move | הפונקציה מזיזה את הנקודה במרחב, בכל ציר בערך שהוזן. |
| SetPointCoordinates(); | Axis axis, double coordinate\_value | הפונקציה מגדירה את הקורדינטה של הנקודה בציר המבוקש. |
| SetPointCoordinates(); | double x\_coordinate double y\_coordinate double z\_coordinate | הפונקציה מגדירה את הקורדינטות של הנקודה מחדש. |
| ScalePoint(); | Axis axis, double scale\_ratio | הפונקציה משנה את קנה המידה של הנקודה בציר המבוקש. |
| ScalePoint(); | double x\_ratio, double y\_ratio, double z\_ratio | הפונקציה משנה את קנה המידה של הנקודה בכל ציר לפי קנה המידה שהוגדר לכל ציר. |
| RotatePoint(); | Axis axis, double rotate\_value | הפונקציה מסובבת נקודה סביב הציר המבוקש בערך שהוזן. |
| SetPointColor(); | Color color\_component, double color\_value | הפונקציה מגדירה את הערך של הצבע המבוקש לערך שהוזן. |
| SetPointColor(); | double red\_component, double green\_component, double blue\_component | הפונקציה מגדירה את ערכי כל אחד מן הצבעים של הנקודה. |
| ClonePoint(); |  | הפונקציה יוצרת נקודה נוספת עם אותם ערכים של הנקודה הנוכחית. |
| GetCoordinate(); | Axis axis | הפונקציה מחזירה את הקורדינטות של הנקודה בציר המבוקש. |
| GetColor(); | Color color\_component | הפונקציה מחזירה את ערך הצבע המבוקש של הנקודה. |
| DrawPoint(); |  | הפונקציה מציירת את הנקודה. |
| MovePointX(); | double x\_move | הפונקציה מזיזה את הנקודה בציר הx לפי הערך שהוזן. |
| MovePointY(); | double y\_move | הפונקציה מזיזה את הנקודה בציר הy לפי הערך שהוזן. |
| MovePointZ(); | double z\_move | הפונקציה מזיזה את הנקודה בציר הz לפי הערך שהוזן. |
| SetPointX(); | double x\_coordinate | הפונקציה משנה את ערך הx לערך שהוזן. |
| SetPointY(); | double y\_coordinate | הפונקציה משנה את ערך הyלערך שהוזן. |
| SetPointZ(); | double z\_coordinate | הפונקציה משנה את ערך הz לערך שהוזן. |
| RotatePointX(); | double x\_rotate | הפונקציה מסובבת את הנקודה סביב ציר הx בערך שהוזן. |
| RotatePointY(); | double y\_rotate | הפונקציה מסובבת את הנקודה סביב ציר הy בערך שהוזן. |
| RotatePointZ(); | double z\_rotate | הפונקציה מסובבת את הנקודה סביב ציר הz בערך שהוזן. |
| SetPointRed(); | double red\_component | הפונקציה מגדירה את ערך הצבע האדום של הנקודה לפי הערך שהוזן. |
| SetPointGreen(); | double green\_component | הפונקציה מגדירה את ערך הצבע הירוק של הנקודה לפי הערך שהוזן. |
| SetPointBlue(); | double blue\_component | הפונקציה מגדירה את ערך הצבע הכחול של הנקודה לפי הערך שהוזן. |
| GetX(); |  | הפונקציה מחזירה את ערך הx של הנקודה. |
| GetY(); |  | הפונקציה מחזירה את ערך הy של הנקודה. |
| GetZ(); |  | הפונקציה מחזירה את ערך הz של הנקודה. |
| GetRed(); |  | הפונקציה מחזירה את ערך הצבע האדום של הנקודה. |
| GetGreen(); |  | הפונקציה מחזירה את ערך הצבע הירוק של הנקודה. |
| GetBlue(); |  | הפונקציה מחזירה את ערך הצבע הכחול של הנקודה. |
| המשתנים במחלקה: | double \_x\_coordiante double \_y\_coordinate double \_z\_coordinate double \_red\_component double \_green\_component double \_blue\_component | הקורדינטות והצבעים של הנקודה. |
| המחלקה יורשת מCPoint ובעצם בנויה ממחלקה של Points . |  | **CPolygon:** |
| CPolygon(); |  | הבנאי הבסיסי – יוצר הופעה חדשה של שתי נקודות מאופסות אותן הוא מכניס למערך. |
| CPolygon(); | int num\_of\_points | בנאי נוסף – יוצר מערך נקודות בגודל שהוזן ומאפס את כל הנקודות. |
| CPolygon(); | Const CPolygon &Polygon | בנאי העתקה – מקבל עצם ויוצר אובייקט חדש על פי העצם. |
| SetNumberOfPoints(); | int num\_of\_points | הפונקציה מגדירה את מספר הנקודות בפוליגון ומאפסת אותן. |
| SetPolygonColor(); | Color color\_component,  double color\_value | הפונקציה משנה את ערך הצבע המבוקש של כל הנקודות על הפוליגון. |
| SetPolygonColor(); | double red\_component, double green\_component, double blue\_component | הפונקציה משנה את כל ערכי הצבעים של כל הנקודות על הפוליגון. |
| DuplicatePolygon(); | CPolygon polygon | הפונקציה משנה את מערך הנקודות של הפוליגון למערך הנקודות של הפוליגון שקיבלה. |
| SetOnePoint(); | int point\_index, CPoint point | הפונקציה מגדירה את הנקודה המבוקשת בפוליגון לנקודה שקיבלה הפונקציה. |
| SetOnePointCoordiantes(); | int point\_index, double x\_coordinate double y\_coordinate double z\_coordinate | הפונקציה מגדירה את הקורדינטות של הנקודה המבוקשת בפוליגון. |
| SetOnePointColor(); | int point\_index, Color color\_component, double color\_value | הפונקציה משנה את ערך הצבע המבוקש בערך שקיבלה לנקודה המבוקשת. |
| SetOnePointColor(); | int point\_index, double red\_component, double green\_component, double blue\_component | הפונקציה משנה את ערכי כל הצבעים לנקודה המבוקשת. |
| MovePolygon(); | Axis axis, double move\_value | הפונקציה מזיזה את הפוליגון על הציר המבוקש בערך שקיבלה. |
| MovePolygon(); | double x\_move, double y\_move, double z\_move | הפונקציה מזיזה את הפוליגון בכל ציר בערך שקיבלה לו. |
| MovePolygon(); | CPoint point\_move | הפונקציה מזיזה את כל הנקודות בפוליגון בערכי הנקודה שקיבלה. |
| MovePolygonCenter(); | Axis axis, double coordinate\_value | הפונקציה מזיזה את נקודת מרכז הפוליגון בציר המבוקש בערך שקיבלה. |
| MovePolygonCenter(); | double x\_coordinate double y\_coordinate double z\_coordinate | הפונקציה מזיזה את נקודת מרכז הפוליגון בכל הצירים בערכים שקיבלה לכל ציר. |
| MovePolygonCenter(); | Cpoint point\_coordinate | הפונקציה מזיזה את נקודת מרכז הפוליגון לנקודה שקיבלה. |
| ScalePolygon(); | Axis axis,  double scale\_ratio | הפונקציה משנה את קנה המידה של הפוליגון בציר הנבחר. |
| ScalePolygon(); | double x\_ratio double y\_ratio double z\_ratio | הפונקציה משנה את קנה המידה של הפוליגון בכל הצירים בערך שקיבלה לכל ציר. |
| ScaleSelfPolygon(); | Axis axis, double scale\_ratio | הפונקציה משנה את קנה המידה של הפוליגון בציר הנבחר, ביחס למרכז הפוליגון. |
| ScaleSelfPolygon(); | double x\_ratio double y\_ratio double z\_ratio | הפונקציה משנה את קנה המידה של הפוליגון בכל ציר בערך שקיבלה לכל ציר, ביחס למרכז הפוליגון. |
| RotatePolygon(); | Axis axis, double rotate\_value | הפונקציה מסובבת את הפוליגון סביב הציר המבוקש בערך שקיבלה. |
| RotateSelfPolygon(); | Axis axis, double rotate\_value | הפונקציה מסובבת את הפוליגון סביב הציר המבוקש בערך שקיבלה, ביחס למרכז הפוליגון. |
| DrawPolygon(); |  | הפונקציה מציירת את הפוליגון. |
| GetPoint(); | int point\_index | הפונקציה מחזירה את הנקודה על הפוליגון באינדקס המבוקש מתוך מערך הנקודות של הפוליגון. |
| GetPolygonCenter(); |  | הפונקציה מחזירה את מרכז הפוליגון. |
| GetNumberOfPoints(); |  | הפונקציה מחזירה את מספר הנקודות בפוליגון. |
| המשתנים במחלקה: | int \_num\_of\_points CPoint\* \_points\_array |  |
|  |  | **CModel:** |
| CModel(); |  | הבנאי הבסיסי – יוצר הופעה חדשה שבה יש שני פוליגונים מאופסים במערך הפוליגונים של המודל. |
| CModel(); | int num\_of\_polygons | בנאי נוסף – יוצר הופעה חדשה שבה מספר הפוליגונים במערך של המודל הוא המשתנה שקיבל הבנאי (מאופסים). |
| CModel(); | Const CModel &model | בנאי העתקה – מקבל עצם מסוג מודל ובונה את המודל החדש לפי המודל שקיבל. |
| SetNumberOfPolygons(); | int num\_of\_polygons | הפונקציה מגדירה את מספר הפוליגונים שיש במערך הפוליגונים של המודל. |
| SetModelColor(); | Color color\_component,  double color\_value | הפונקציה משנה את ערך הצבע המבוקש של כל הפוליגונים במודל. |
| SetModelColor(); | double red\_component, double green\_component, double blue\_component | הפונקציה משנה את כל ערכי הצבעים של כל הפוליגונים על המודל. |
| SetOnePolygon(); | int polygon\_index, CPolygon polygon | הפונקציה מגדירה את הפוליגון באינדקס המבוקש להיות הפוליגון שקיבלה. |
| SetOnePolygonColor(); | int polygon\_index, Color color\_component,  double color\_value | הפונקציה משנה את אחד מערכי הצבעים(המבוקש) של אחד מן הפוליגונים במערך של המודל בערך שקיבלה. |
| SetOnePolygonColor(); | int polygon\_index, double red\_component, double green\_component, double blue\_component | הפונקציה משנה את כל ערכי הצבעים של אחד מן הפוליגונים במערך של המודל. |
| MoveModel(); | Axis axis, double move\_value | הפונקציה מזיזה את המודל בציר המבוקש, בערך שקיבלה. |
| MoveModel(); | double x\_move, double y\_move, double z\_move | הפונקציה מזיזה את המודל בכל ציר לפי הערך שקיבלה לו. |
| MoveModel(); | CPoint point\_move | הפונקציה מזיזה את המודל על פי הקורדינטות של הנקודה שקיבלה. |
| MoveModelCenter(); | Axis axis, double coordinate\_value | הפונקציה מזיזה את מרכז המודל בציר המבוקש בערך שקיבלה. |
| MoveModelCenter(); | double x\_coordinate double y\_coordinate double z\_coordinate | הפונקציה מזיזה את מרכז המודל בכל ציר לפי הערך שקיבלה לו. |
| MoveModelCenter(); | Cpoint point\_coordinate | הפונקציה מזיזה את מרכז המודל לנקודה שקיבלה. |
| ScaleModel(); | Axis axis, double scale\_ratio | הפונקציה משנה את קנה המידה של המודל בציר המבוקש בערך שקיבלה. |
| ScaleModel(); | double x\_ratio double y\_ratio double z\_ratio | הפונקציה משנה את קנה המידה של המודל בכל ציר לפי הערך שקיבלה לו. |
| ScaleSelfModel(); | Axis axis, double scale\_ratio | הפונקציה משנה את קנה המידה של המודל בציר המבוקש בערך שקיבלה, ביחס למרכז המודל. |
| ScaleSelfModel(); | double x\_ratio double y\_ratio double z\_ratio | הפונקציה משנה את קנה המידה של המודל בכל ציר לפי הערך שקיבלה לו, ביחס למרכז המודל. |
| RotateModel(); | Axis axis, double rotate\_value | הפונקציה מסובבת את המודל סביב הציר המבוקש בערך שקיבלה. |
| RotateSelfModel(); | Axis axis, double rotate\_value | הפונקציה מסובבת את המודל סביב הציר המבוקש בערך שקיבלה, ביחס למרכז המודל. |
| DrawModel(); |  | הפונקציה מציירת את המודל. |
| GetPolygon(); | int polygon\_index | הפונקציה מחזירה את הפוליגון הנמצא בערך המבוקש במערך הפוליגונים של המודל. |
| GetModelCenter(); |  | הפונקציה מחזירה את מרכז המודל. |
| GetNumberOfPolygons(); |  | הפונקציה מחזירה את מספר הפוליגונים במודל. |
| המשתנים במחלקה: | Int \_num\_of\_polygons CPolygon\* \_polygons\_array | מספר הפוליגונים במודל. מערך הפוליגונים.  **CBall:** |
| יורש מCModel (בנוי מפוליגונים) |  |  |
| CBall(); |  | הבנאי הבסיסי – יוצר הופעה חדשה של כדור כאשר המשתנים של המחלקה מוגדרים מראש. |
| CBall(); | int num\_of\_fi\_values, int num\_of\_theta\_values | בנאי נוסף – יוצר הופעה חדשה של כדור עם הערכים שקיבל הבנאי. |
| SetBallNumOfElements(); | int num\_of\_fi\_values, int num\_of\_theta\_values | הפונקציה משנה את הערכים של הזוויות fi ו theta אשר בעצם בונות את הכדור. |
| SetBallColors(); | double red\_component\_min, double red\_component\_max, double green\_component\_min, double green\_component\_max, double blue\_component\_min, double blue\_component\_max | הפונקציה משנה את צבע הכדור לפי הצבעים שקבענו, על מנת שייראה תלת מימדי הצבעים בחלקים שונים בכדור ינועו בין הערכים שקלטנו. |
| GetNewColorComponent(); | int fi\_index, int theta\_index, int fi\_weight, int theta\_weight | הפונקציה מחזירה מספר בין 1 ל-0 אשר יקבע את הצבע של כל חלק בכדור. |

**תרשים זרימה:**

תהליך יריית הכדור:



**סיכום אישי:  
צדדים בפרויקט:**

חיובי:

- המשחק מאתגר מאוד.

- המשחק קל להבנה.

- גרפיקה פשוטה ולא מבלבלת.

-הקוד יעיל מאוד, כמעט כל הפונקציות והערכים נמצאים במחלקות משלהם.

שלילי:

- יש רק 3 שלבים במשחק.

- קשה מאוד לנצח את המשחק.

אילו היה לי יותר זמן:

- הייתי מוסיף עוד משחקיות (עוד שלבים עם רמות קושי גבוהות יותר).

- משפר את הגרפיקה.

מה למדתי:

**גרפיקה:** בניית מודלים, הנפשה בעזרת קוד.

**תכנות:** כשכתבתי את הפרויקט בעזרת המנחה שלי דור שהסביר לי איך לייעל קוד בצורה טובה יותר, למדתי לייעל את הקוד שלי בצורה טובה יותר, וכך עשיתי בפרויקט.

**פרויקט:** הבנתי איך בונים פרויקט מ0 , שלבי התכנון, בניית הקוד, ייעולו, והגשתו.

**ארגון העבודה:** לאחר הפרויקט למדתי איך לארגן את הזמן שלי ואת העבודה שלי בצורה טובה יותר.

דרך מימוש הפרויקט:

בנינו מחלקות החל מPoint ואז לPolygon ואז לModel לאחר מכן החלנו לבנות את האובייקטים שמשתתפים בפרויקט ולאחר מכן הנפשנו והכנו את המשחקיות וייעלנו את הקוד.

רקע מתמטי:

שימוש בנוסחאות מתמטיות כגון נוסחאת דיסטנס, שטח והיקף מעגל; שימוש בערך פאי.  
שימוש בספריית "math.h" שבה טמונים נוסחאות sin או cos או tan וכו'.

**קוד התכנית:**קבצי Header (h.):

**Bullet.h:**  
#pragma once

#include "CBall.h"

#include "Target.h"

#include "GameState.h"

class Bullet :

public CBall

{

public:

Bullet();

void BulletShot();

void BulletComeBack();

void BulletHitCheck(GameState &gs1, Target &target);  
};

**CBall.h:**  
#pragma once

#include "CModel.h"

/// Create colorful ball, with radius in (x,y,z)=1 and centered at (0,0,0)

class CBall : public CModel

{

public:

#pragma region CONSTRUCTORs

CBall(void);

CBall(int num\_of\_fi\_values, int num\_of\_theta\_values);

#pragma endregion

#pragma region Set ball properties

void SetBallNumOfElements(int num\_of\_fi\_values, int num\_of\_theta\_values);

void SetBallColors(double red\_component\_min, double red\_component\_max, double green\_component\_min, double green\_component\_max, double blue\_component\_min, double blue\_component\_max);

#pragma endregion

private:

double GetNewColorComponent(int fi\_index, int theta\_index, int fi\_weight, int theta\_weight);

#pragma region Private variables

int \_num\_of\_fi\_values;

int \_num\_of\_theta\_values;

#pragma endregion

};

**CCircle.h:**  
#pragma once

#include "CPolygon.h"

/// Create black circle, with radius in (x,y) = 1 and with center at (0,0,0)

class CCircle : public CPolygon

{

public:

#pragma region CONSTRUCTORs

CCircle(void);

CCircle(int num\_of\_points);

#pragma endregion

void SetNumberOfPoints(int num\_of\_points);

};

**CGameHandler.h:**

#pragma once

class CGameHandler

{

public:

//the beginning

CGameHandler(){};

void StartOfGame();

//on each frame:

void DrawScene();

void LogicPart();

void WriteText();

void KeyPressed(char key);

void SpecialKeyPressed(char key);

//getting backgroundColor

float GetBeckgroundColorR();

float GetBeckgroundColorG();

float GetBeckgroundColorB();

private:

//other procedures

void ChangeBackgroundColors(float R, float G, float B);

//internal variables:

float BGColor\_R,BGColor\_G,BGColor\_B; //BackGround colors.

int font,def\_font; //determine the font to the text written.

char\* WrongKeyText;

};

**CModel.h:**  
#pragma once

#include "CPoint.h"

#include "CPolygon.h"

class CModel

{

public:

#pragma region CONSTRUCTORs & DESTRUCTOR

CModel();

CModel(int num\_of\_polygons);

CModel(const CModel& model);

~CModel();

#pragma endregion

#pragma region Setting model properties

void SetNumberOfPolygons(int num\_of\_polygons);

void SetModelColor(Color color\_component, double color\_value);

void SetModelColor(double red\_component, double green\_component, double blue\_component);

#pragma endregion

#pragma region Setting one polygon properties

void SetOnePolygon(int polygon\_index, CPolygon polygon);

void SetOnePolygonColor(int polygon\_index, Color color\_component, double color\_value);

void SetOnePolygonColor(int polygon\_index, double red\_component, double green\_component, double blue\_component);

#pragma endregion

#pragma region Move model functions

void MoveModelTo(double x\_Coordinate, double y\_Coordinate, double z\_Coordinate);

void MoveModel(Axis axis, double move\_value);

void MoveModel(double x\_move, double y\_move,double z\_move);

void MoveModel(CPoint point\_move);

void MoveModelCenter(Axis axis, double coordinate\_value);

void MoveModelCenter(double x\_coordinate, double y\_coordinate,double z\_coordinate);

void MoveModelCenter(CPoint point\_coordinate);

#pragma endregion

#pragma region Scale model functions

void ScaleModel(Axis axis, double scale\_ratio);

void ScaleModel(double x\_ratio, double y\_ratio, double z\_ratio);

void ScaleSelfModel(Axis axis, double scale\_ratio);

void ScaleSelfModel(double x\_ratio, double y\_ratio, double z\_ratio);

#pragma endregion

#pragma region Rotate model functions

void RotateModel(Axis axis, double rotate\_value);

void RotateSelfModel(Axis axis, double rotate\_value);

#pragma endregion

#pragma region Draw model function

void DrawModel();

#pragma endregion

#pragma region Get functions

CPolygon GetPolygon(int polygon\_index);

CPoint GetModelCenter();

int GetNumberOfPolygons();

#pragma endregion

protected:

#pragma region Private variables

int \_num\_of\_polygons;

CPolygon\* \_polygons\_array;

#pragma endregion

};

**CMyOpenGLInit.h:**#pragma once

class CMyOpenGLInit

{

public:

CMyOpenGLInit(int argc, char\* argv[]);

void CMyOpenGLInit::InitParameters();

};  
**Gun.h:**#pragma once

#include "CModel.h"

class Gun : public CModel

{

public:

Gun(void);

};  
**Target.h:**  
#if !defined( TARGET\_H )

#define TARGET\_H

#pragma once

#include "CCircle.h"

#include "CModel.h"

#include "GameState.h"

class Target : public CModel

{

public:

Target();

void TargetMovement(GameState &gs1);

~Target();

};

#endif

**CPoint.h:**#pragma once

#pragma region ENUMs

void DrawPoint();

enum Axis

{

X\_AXIS = 1,

Y\_AXIS = 2,

Z\_AXIS = 3

};

enum Color

{

RED\_COMPONENT = 1,

GREEN\_COMPONENT = 2,

BLUE\_COMPONENT = 3

};

#pragma endregion

class CPoint

{

public:

#pragma region CONSTRUCTORs

CPoint();

CPoint(double x\_coordinate, double y\_coordinate, double z\_coordinate);

CPoint(double x\_coordinate, double y\_coordinate, double z\_coordinate, double red\_component, double green\_component, double blue\_component);

#pragma endregion

#pragma region Move point functions

void MovePointTo(double x\_Coordinate, double y\_Coordinate, double z\_Coordinate);

void MovePoint(Axis axis, double move\_value);

void MovePoint(double x\_move, double y\_move, double z\_move);

void MovePoint(CPoint point\_move);

#pragma endregion

#pragma region Set point location functions

void SetPointCoordinate(Axis axis, double coordinate\_value);

void SetPointCoordinates(double x\_coordinate, double y\_coordinate, double z\_coordinate);

#pragma endregion

#pragma region Scale point functions

void ScalePoint(Axis axis, double scale\_ratio);

void ScalePoint(double x\_ratio, double y\_ratio, double z\_ratio);

#pragma endregion

#pragma region Rotate point functions

void RotatePoint(Axis axis, double rotate\_value);

#pragma endregion

#pragma region Set point color functions

void SetPointColor(Color color\_component, double color\_value);

void SetPointColor(double red\_component, double green\_component, double blue\_component);

#pragma endregion

#pragma region Get functions

CPoint ClonePoint();

double GetCoordinate(Axis axis);

double GetColor(Color color\_component);

#pragma endregion

#pragma region Draw function

void DrawPoint();

#pragma endregion

private:

#pragma region Private variables

double \_x\_coordinate;

double \_y\_coordinate;

double \_z\_coordinate;

double \_red\_component;

double \_green\_component;

double \_blue\_component;

#pragma endregion

#pragma region Move point functions

void MovePointX(double x\_move);

void MovePointY(double y\_move);

void MovePointZ(double z\_move);

#pragma endregion

#pragma region Set point location functions

void SetPointX(double x\_coordinate);

void SetPointY(double y\_coordinate);

void SetPointZ(double z\_coordinate);

#pragma endregion

#pragma region Rotate point functions

void RotatePointX(double x\_rotate);

void RotatePointY(double y\_rotate);

void RotatePointZ(double z\_rotate);

#pragma endregion

#pragma region Set point color functions

void SetPointRed(double red\_component);

void SetPointGreen(double green\_component);

void SetPointBlue(double blue\_component);

#pragma endregion

#pragma region Get functions

double GetX();

double GetY();

double GetZ();

double GetRed();

double GetGreen();

double GetBlue();

#pragma endregion

};

**CPolygon.h:**#pragma once

#include "CPoint.h"

class CPolygon

{

public:

#pragma region CONSTRUCTORs & DESTRUCTOR

CPolygon();

CPolygon(int num\_of\_points);

CPolygon(const CPolygon& polygon);

~CPolygon();

#pragma endregion

#pragma region Setting polygon properties

void SetNumberOfPoints(int num\_of\_points);

void SetPolygonColor(Color color\_component, double color\_value);

void SetPolygonColor(double red\_component, double green\_component, double blue\_component);

void DuplicatePolygon(CPolygon polygon);

#pragma endregion

#pragma region Setting one point properties

void SetOnePoint(int point\_index, CPoint point);

void SetOnePointCoordinates(int point\_index, double x\_coordinate, double y\_coordinate, double z\_coordinate);

void SetOnePointColor(int point\_index, Color color\_component, double color\_value);

void SetOnePointColor(int point\_index, double red\_component, double green\_component, double blue\_component);

#pragma endregion

#pragma region Move polygon functions

void MovePolygonTo(double x\_Coordinate, double y\_Coordinate, double z\_Coordinate);

void MovePolygon(Axis axis, double move\_value);

void MovePolygon(double x\_move, double y\_move,double z\_move);

void MovePolygon(CPoint point\_move);

void MovePolygonCenter(Axis axis, double coordinate\_value);

void MovePolygonCenter(double x\_coordinate, double y\_coordinate,double z\_coordinate);

void MovePolygonCenter(CPoint point\_coordinate);

#pragma endregion

#pragma region Scale polygon functions

void ScalePolygon(Axis axis, double scale\_ratio);

void ScalePolygon(double x\_ratio, double y\_ratio, double z\_ratio);

void ScaleSelfPolygon(Axis axis, double scale\_ratio);

void ScaleSelfPolygon(double x\_ratio, double y\_ratio, double z\_ratio);

#pragma endregion

#pragma region Rotate polygon functions

void RotatePolygon(Axis axis, double rotate\_value);

void RotateSelfPolygon(Axis axis, double rotate\_value);

#pragma endregion

#pragma region Draw polygon function

void DrawPolygon();

#pragma endregion

#pragma region Get functions

CPoint GetPoint(int point\_index);

CPoint GetPolygonCenter();

int GetNumberOfPoints();

#pragma endregion

private:

#pragma region Private variables

int \_num\_of\_points;

CPoint\* \_points\_array;

#pragma endregion

};

**GameState.h:**#if !defined( GAMESTATE\_H )

#define GAMESTATE\_H

#pragma once

class GameState

{

public:

GameState();

void Gameover();

void StageChange();

void GameFinished();

void Stage2LivePlus1();

void Stage3LivePlus3();

int GetLives();

void SetLives(int lives);

double GetXChange();

void SetXChange(double x);

double GetGunMove();

void SetGunMove(double gm);

bool GetGameOver();

void SetGameOver(bool GO);

int GetStage();

void SetStage(int St);

int GetLevelNum();

void SetLevelNum(int LN);

double GetXCMulti();

void SetXCMulti(double xCM);

bool GetCongratz();

void SetCongratz(bool C);

bool GetStageUp();

void SetStageUp(bool SU);

bool GetShotMissed();

void SetShotMissed(bool SM);

bool GetBulletHitTarget();

void SetBulletHitTarget(bool BHT);

bool GetBullsEye();

void SetBullsEye(bool BE);

bool GetGameStarted();

void SetGameStarted(bool GS);

bool GetFirstShotBeenShot();

void SetFirstShotBeenShot(bool FSBS);

bool GetGameReboot();

void SetGameReboot(bool GGR);

bool GetBBS();

void SetBBS(bool BULLETBS);

void RESET();

void AdvanceLevel();

void ShootingHandler();

private:

int lives;

double xChange;

double gunMove;

bool GameOver;

int Stage;

int Level\_Number;

double xChange\_Multiplier;

bool Congratz;

bool StageUp;

bool ShotMissed;

bool BulletHitTarget;

bool BullsEye;

bool GameStarted;

bool FirstShotBeenShot;

bool GameReboot;

bool BULLETbeenSHOT;

};

#endif

**ספרייה:Glut.h**  
#ifndef \_\_glut\_h\_\_

#define \_\_glut\_h\_\_

/\* Copyright (c) Mark J. Kilgard, 1994, 1995, 1996, 1998. \*/

#if defined(\_WIN32)

# if 0

# define WIN32\_LEAN\_AND\_MEAN

# include <windows.h>

# else

# ifndef APIENTRY

# define GLUT\_APIENTRY\_DEFINED

# if (\_MSC\_VER >= 800) || defined(\_STDCALL\_SUPPORTED) || defined(\_\_BORLANDC\_\_) || defined(\_\_LCC\_\_)

# define APIENTRY \_\_stdcall

# else

# define APIENTRY

# endif

# endif

# ifndef CALLBACK

# if (defined(\_M\_MRX000) || defined(\_M\_IX86) || defined(\_M\_ALPHA) || defined(\_M\_PPC)) && !defined(MIDL\_PASS) || defined(\_\_LCC\_\_)

# define CALLBACK \_\_stdcall

# else

# define CALLBACK

# endif

# endif

# if defined( \_\_LCC\_\_ )

# undef WINGDIAPI

# define WINGDIAPI \_\_stdcall

# else

# ifndef WINGDIAPI

# define GLUT\_WINGDIAPI\_DEFINED

# define WINGDIAPI \_\_declspec(dllimport)

# endif

# endif

# ifndef \_WCHAR\_T\_DEFINED

typedef unsigned short wchar\_t;

# define \_WCHAR\_T\_DEFINED

# endif

# endif

# if !defined(GLUT\_BUILDING\_LIB) && !defined(GLUT\_NO\_LIB\_PRAGMA)

# pragma comment (lib, "winmm.lib")

# ifdef GLUT\_USE\_SGI\_OPENGL

# pragma comment (lib, "opengl.lib")

# pragma comment (lib, "glu.lib")

# pragma comment (lib, "glut.lib")

# else

# pragma comment (lib, "opengl32.lib")

# pragma comment (lib, "glu32.lib")

# pragma comment (lib, "glut32.lib")

# endif

# endif

# ifndef GLUT\_NO\_WARNING\_DISABLE

# pragma warning (disable:4244)

# pragma warning (disable:4305)

# endif

# if !defined(\_MSC\_VER) && !defined(\_\_cdecl)

# define \_\_cdecl

# define GLUT\_DEFINED\_\_\_CDECL

# endif

# ifndef \_CRTIMP

# ifdef \_NTSDK

# define \_CRTIMP

# else

# ifdef \_DLL

# define \_CRTIMP \_\_declspec(dllimport)

# else

# define \_CRTIMP

# endif

# endif

# define GLUT\_DEFINED\_\_CRTIMP

# endif

# ifdef GLUT\_BUILDING\_LIB

# define GLUTAPI \_\_declspec(dllexport)

# else

# ifdef \_DLL

# define GLUTAPI \_\_declspec(dllimport)

# else

# define GLUTAPI extern

# endif

# endif

# define GLUTCALLBACK \_\_cdecl

#endif

#include <GL/gl.h>

#include <GL/glu.h>

#ifdef \_\_cplusplus

extern "C" {

#endif

#if defined(\_WIN32)

# ifndef GLUT\_BUILDING\_LIB

extern \_CRTIMP void \_\_cdecl exit(int);

# endif

#else

# define APIENTRY

# define GLUT\_APIENTRY\_DEFINED

# define CALLBACK

# define GLUTAPI extern

# define GLUTCALLBACK

extern void exit(int);

#endif

#ifndef GLUT\_API\_VERSION

#define GLUT\_API\_VERSION 3

#endif

#ifndef GLUT\_XLIB\_IMPLEMENTATION

#define GLUT\_XLIB\_IMPLEMENTATION 15

#endif

#define GLUT\_RGB 0

#define GLUT\_RGBA GLUT\_RGB

#define GLUT\_INDEX 1

#define GLUT\_SINGLE 0

#define GLUT\_DOUBLE 2

#define GLUT\_ACCUM 4

#define GLUT\_ALPHA 8

#define GLUT\_DEPTH 16

#define GLUT\_STENCIL 32

#if (GLUT\_API\_VERSION >= 2)

#define GLUT\_MULTISAMPLE 128

#define GLUT\_STEREO 256

#endif

#if (GLUT\_API\_VERSION >= 3)

#define GLUT\_LUMINANCE 512

#endif

#define GLUT\_LEFT\_BUTTON 0

#define GLUT\_MIDDLE\_BUTTON 1

#define GLUT\_RIGHT\_BUTTON 2

#define GLUT\_DOWN 0

#define GLUT\_UP 1

#if (GLUT\_API\_VERSION >= 2)

#define GLUT\_KEY\_F1 1

#define GLUT\_KEY\_F2 2

#define GLUT\_KEY\_F3 3

#define GLUT\_KEY\_F4 4

#define GLUT\_KEY\_F5 5

#define GLUT\_KEY\_F6 6

#define GLUT\_KEY\_F7 7

#define GLUT\_KEY\_F8 8

#define GLUT\_KEY\_F9 9

#define GLUT\_KEY\_F10 10

#define GLUT\_KEY\_F11 11

#define GLUT\_KEY\_F12 12

#define GLUT\_KEY\_LEFT 100

#define GLUT\_KEY\_UP 101

#define GLUT\_KEY\_RIGHT 102

#define GLUT\_KEY\_DOWN 103

#define GLUT\_KEY\_PAGE\_UP 104

#define GLUT\_KEY\_PAGE\_DOWN 105

#define GLUT\_KEY\_HOME 106

#define GLUT\_KEY\_END 107

#define GLUT\_KEY\_INSERT 108

#endif

#define GLUT\_LEFT 0

#define GLUT\_ENTERED 1

#define GLUT\_MENU\_NOT\_IN\_USE 0

#define GLUT\_MENU\_IN\_USE 1

#define GLUT\_NOT\_VISIBLE 0

#define GLUT\_VISIBLE 1

#define GLUT\_HIDDEN 0

#define GLUT\_FULLY\_RETAINED 1

#define GLUT\_PARTIALLY\_RETAINED 2

#define GLUT\_FULLY\_COVERED 3

#define GLUT\_RED 0

#define GLUT\_GREEN 1

#define GLUT\_BLUE 2

#if defined(\_WIN32)

#define GLUT\_STROKE\_ROMAN ((void\*)0)

#define GLUT\_STROKE\_MONO\_ROMAN ((void\*)1)

#define GLUT\_BITMAP\_9\_BY\_15 ((void\*)2)

#define GLUT\_BITMAP\_8\_BY\_13 ((void\*)3)

#define GLUT\_BITMAP\_TIMES\_ROMAN\_10 ((void\*)4)

#define GLUT\_BITMAP\_TIMES\_ROMAN\_24 ((void\*)5)

#if (GLUT\_API\_VERSION >= 3)

#define GLUT\_BITMAP\_HELVETICA\_10 ((void\*)6)

#define GLUT\_BITMAP\_HELVETICA\_12 ((void\*)7)

#define GLUT\_BITMAP\_HELVETICA\_18 ((void\*)8)

#endif

#else

GLUTAPI void \*glutStrokeRoman;

GLUTAPI void \*glutStrokeMonoRoman;

#define GLUT\_STROKE\_ROMAN (&glutStrokeRoman)

#define GLUT\_STROKE\_MONO\_ROMAN (&glutStrokeMonoRoman)

GLUTAPI void \*glutBitmap9By15;

GLUTAPI void \*glutBitmap8By13;

GLUTAPI void \*glutBitmapTimesRoman10;

GLUTAPI void \*glutBitmapTimesRoman24;

GLUTAPI void \*glutBitmapHelvetica10;

GLUTAPI void \*glutBitmapHelvetica12;

GLUTAPI void \*glutBitmapHelvetica18;

#define GLUT\_BITMAP\_9\_BY\_15 (&glutBitmap9By15)

#define GLUT\_BITMAP\_8\_BY\_13 (&glutBitmap8By13)

#define GLUT\_BITMAP\_TIMES\_ROMAN\_10 (&glutBitmapTimesRoman10)

#define GLUT\_BITMAP\_TIMES\_ROMAN\_24 (&glutBitmapTimesRoman24)

#if (GLUT\_API\_VERSION >= 3)

#define GLUT\_BITMAP\_HELVETICA\_10 (&glutBitmapHelvetica10)

#define GLUT\_BITMAP\_HELVETICA\_12 (&glutBitmapHelvetica12)

#define GLUT\_BITMAP\_HELVETICA\_18 (&glutBitmapHelvetica18)

#endif

#endif

#define GLUT\_WINDOW\_X ((GLenum) 100)

#define GLUT\_WINDOW\_Y ((GLenum) 101)

#define GLUT\_WINDOW\_WIDTH ((GLenum) 102)

#define GLUT\_WINDOW\_HEIGHT ((GLenum) 103)

#define GLUT\_WINDOW\_BUFFER\_SIZE ((GLenum) 104)

#define GLUT\_WINDOW\_STENCIL\_SIZE ((GLenum) 105)

#define GLUT\_WINDOW\_DEPTH\_SIZE ((GLenum) 106)

#define GLUT\_WINDOW\_RED\_SIZE ((GLenum) 107)

#define GLUT\_WINDOW\_GREEN\_SIZE ((GLenum) 108)

#define GLUT\_WINDOW\_BLUE\_SIZE ((GLenum) 109)

#define GLUT\_WINDOW\_ALPHA\_SIZE ((GLenum) 110)

#define GLUT\_WINDOW\_ACCUM\_RED\_SIZE ((GLenum) 111)

#define GLUT\_WINDOW\_ACCUM\_GREEN\_SIZE ((GLenum) 112)

#define GLUT\_WINDOW\_ACCUM\_BLUE\_SIZE ((GLenum) 113)

#define GLUT\_WINDOW\_ACCUM\_ALPHA\_SIZE ((GLenum) 114)

#define GLUT\_WINDOW\_DOUBLEBUFFER ((GLenum) 115)

#define GLUT\_WINDOW\_RGBA ((GLenum) 116)

#define GLUT\_WINDOW\_PARENT ((GLenum) 117)

#define GLUT\_WINDOW\_NUM\_CHILDREN ((GLenum) 118)

#define GLUT\_WINDOW\_COLORMAP\_SIZE ((GLenum) 119)

#if (GLUT\_API\_VERSION >= 2)

#define GLUT\_WINDOW\_NUM\_SAMPLES ((GLenum) 120)

#define GLUT\_WINDOW\_STEREO ((GLenum) 121)

#endif

#if (GLUT\_API\_VERSION >= 3)

#define GLUT\_WINDOW\_CURSOR ((GLenum) 122)

#endif

#define GLUT\_SCREEN\_WIDTH ((GLenum) 200)

#define GLUT\_SCREEN\_HEIGHT ((GLenum) 201)

#define GLUT\_SCREEN\_WIDTH\_MM ((GLenum) 202)

#define GLUT\_SCREEN\_HEIGHT\_MM ((GLenum) 203)

#define GLUT\_MENU\_NUM\_ITEMS ((GLenum) 300)

#define GLUT\_DISPLAY\_MODE\_POSSIBLE ((GLenum) 400)

#define GLUT\_INIT\_WINDOW\_X ((GLenum) 500)

#define GLUT\_INIT\_WINDOW\_Y ((GLenum) 501)

#define GLUT\_INIT\_WINDOW\_WIDTH ((GLenum) 502)

#define GLUT\_INIT\_WINDOW\_HEIGHT ((GLenum) 503)

#define GLUT\_INIT\_DISPLAY\_MODE ((GLenum) 504)

#if (GLUT\_API\_VERSION >= 2)

#define GLUT\_ELAPSED\_TIME ((GLenum) 700)

#endif

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 13)

#define GLUT\_WINDOW\_FORMAT\_ID ((GLenum) 123)

#endif

#if (GLUT\_API\_VERSION >= 2)

#define GLUT\_HAS\_KEYBOARD ((GLenum) 600)

#define GLUT\_HAS\_MOUSE ((GLenum) 601)

#define GLUT\_HAS\_SPACEBALL ((GLenum) 602)

#define GLUT\_HAS\_DIAL\_AND\_BUTTON\_BOX ((GLenum) 603)

#define GLUT\_HAS\_TABLET ((GLenum) 604)

#define GLUT\_NUM\_MOUSE\_BUTTONS ((GLenum) 605)

#define GLUT\_NUM\_SPACEBALL\_BUTTONS ((GLenum) 606)

#define GLUT\_NUM\_BUTTON\_BOX\_BUTTONS ((GLenum) 607)

#define GLUT\_NUM\_DIALS ((GLenum) 608)

#define GLUT\_NUM\_TABLET\_BUTTONS ((GLenum) 609)

#endif

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 13)

#define GLUT\_DEVICE\_IGNORE\_KEY\_REPEAT ((GLenum) 610)

#define GLUT\_DEVICE\_KEY\_REPEAT ((GLenum) 611)

#define GLUT\_HAS\_JOYSTICK ((GLenum) 612)

#define GLUT\_OWNS\_JOYSTICK ((GLenum) 613)

#define GLUT\_JOYSTICK\_BUTTONS ((GLenum) 614)

#define GLUT\_JOYSTICK\_AXES ((GLenum) 615)

#define GLUT\_JOYSTICK\_POLL\_RATE ((GLenum) 616)

#endif

#if (GLUT\_API\_VERSION >= 3)

#define GLUT\_OVERLAY\_POSSIBLE ((GLenum) 800)

#define GLUT\_LAYER\_IN\_USE ((GLenum) 801)

#define GLUT\_HAS\_OVERLAY ((GLenum) 802)

#define GLUT\_TRANSPARENT\_INDEX ((GLenum) 803)

#define GLUT\_NORMAL\_DAMAGED ((GLenum) 804)

#define GLUT\_OVERLAY\_DAMAGED ((GLenum) 805)

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 9)

#define GLUT\_VIDEO\_RESIZE\_POSSIBLE ((GLenum) 900)

#define GLUT\_VIDEO\_RESIZE\_IN\_USE ((GLenum) 901)

#define GLUT\_VIDEO\_RESIZE\_X\_DELTA ((GLenum) 902)

#define GLUT\_VIDEO\_RESIZE\_Y\_DELTA ((GLenum) 903)

#define GLUT\_VIDEO\_RESIZE\_WIDTH\_DELTA ((GLenum) 904)

#define GLUT\_VIDEO\_RESIZE\_HEIGHT\_DELTA ((GLenum) 905)

#define GLUT\_VIDEO\_RESIZE\_X ((GLenum) 906)

#define GLUT\_VIDEO\_RESIZE\_Y ((GLenum) 907)

#define GLUT\_VIDEO\_RESIZE\_WIDTH ((GLenum) 908)

#define GLUT\_VIDEO\_RESIZE\_HEIGHT ((GLenum) 909)

#endif

#define GLUT\_NORMAL ((GLenum) 0)

#define GLUT\_OVERLAY ((GLenum) 1)

#define GLUT\_ACTIVE\_SHIFT 1

#define GLUT\_ACTIVE\_CTRL 2

#define GLUT\_ACTIVE\_ALT 4

#define GLUT\_CURSOR\_RIGHT\_ARROW 0

#define GLUT\_CURSOR\_LEFT\_ARROW 1

#define GLUT\_CURSOR\_INFO 2

#define GLUT\_CURSOR\_DESTROY 3

#define GLUT\_CURSOR\_HELP 4

#define GLUT\_CURSOR\_CYCLE 5

#define GLUT\_CURSOR\_SPRAY 6

#define GLUT\_CURSOR\_WAIT 7

#define GLUT\_CURSOR\_TEXT 8

#define GLUT\_CURSOR\_CROSSHAIR 9

#define GLUT\_CURSOR\_UP\_DOWN 10

#define GLUT\_CURSOR\_LEFT\_RIGHT 11

#define GLUT\_CURSOR\_TOP\_SIDE 12

#define GLUT\_CURSOR\_BOTTOM\_SIDE 13

#define GLUT\_CURSOR\_LEFT\_SIDE 14

#define GLUT\_CURSOR\_RIGHT\_SIDE 15

#define GLUT\_CURSOR\_TOP\_LEFT\_CORNER 16

#define GLUT\_CURSOR\_TOP\_RIGHT\_CORNER 17

#define GLUT\_CURSOR\_BOTTOM\_RIGHT\_CORNER 18

#define GLUT\_CURSOR\_BOTTOM\_LEFT\_CORNER 19

#define GLUT\_CURSOR\_INHERIT 100

#define GLUT\_CURSOR\_NONE 101

#define GLUT\_CURSOR\_FULL\_CROSSHAIR 102

#endif

GLUTAPI void APIENTRY glutInit(int \*argcp, char \*\*argv);

#if defined(\_WIN32) && !defined(GLUT\_DISABLE\_ATEXIT\_HACK)

GLUTAPI void APIENTRY \_\_glutInitWithExit(int \*argcp, char \*\*argv, void (\_\_cdecl \*exitfunc)(int));

#ifndef GLUT\_BUILDING\_LIB

static void APIENTRY glutInit\_ATEXIT\_HACK(int \*argcp, char \*\*argv) { \_\_glutInitWithExit(argcp, argv, exit); }

#define glutInit glutInit\_ATEXIT\_HACK

#endif

#endif

GLUTAPI void APIENTRY glutInitDisplayMode(unsigned int mode);

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 9)

GLUTAPI void APIENTRY glutInitDisplayString(const char \*string);

#endif

GLUTAPI void APIENTRY glutInitWindowPosition(int x, int y);

GLUTAPI void APIENTRY glutInitWindowSize(int width, int height);

GLUTAPI void APIENTRY glutMainLoop(void);

GLUTAPI int APIENTRY glutCreateWindow(const char \*title);

#if defined(\_WIN32) && !defined(GLUT\_DISABLE\_ATEXIT\_HACK)

GLUTAPI int APIENTRY \_\_glutCreateWindowWithExit(const char \*title, void (\_\_cdecl \*exitfunc)(int));

#ifndef GLUT\_BUILDING\_LIB

static int APIENTRY glutCreateWindow\_ATEXIT\_HACK(const char \*title) { return \_\_glutCreateWindowWithExit(title, exit); }

#define glutCreateWindow glutCreateWindow\_ATEXIT\_HACK

#endif

#endif

GLUTAPI int APIENTRY glutCreateSubWindow(int win, int x, int y, int width, int height);

GLUTAPI void APIENTRY glutDestroyWindow(int win);

GLUTAPI void APIENTRY glutPostRedisplay(void);

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 11)

GLUTAPI void APIENTRY glutPostWindowRedisplay(int win);

#endif

GLUTAPI void APIENTRY glutSwapBuffers(void);

GLUTAPI int APIENTRY glutGetWindow(void);

GLUTAPI void APIENTRY glutSetWindow(int win);

GLUTAPI void APIENTRY glutSetWindowTitle(const char \*title);

GLUTAPI void APIENTRY glutSetIconTitle(const char \*title);

GLUTAPI void APIENTRY glutPositionWindow(int x, int y);

GLUTAPI void APIENTRY glutReshapeWindow(int width, int height);

GLUTAPI void APIENTRY glutPopWindow(void);

GLUTAPI void APIENTRY glutPushWindow(void);

GLUTAPI void APIENTRY glutIconifyWindow(void);

GLUTAPI void APIENTRY glutShowWindow(void);

GLUTAPI void APIENTRY glutHideWindow(void);

#if (GLUT\_API\_VERSION >= 3)

GLUTAPI void APIENTRY glutFullScreen(void);

GLUTAPI void APIENTRY glutSetCursor(int cursor);

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 9)

GLUTAPI void APIENTRY glutWarpPointer(int x, int y);

#endif

GLUTAPI void APIENTRY glutEstablishOverlay(void);

GLUTAPI void APIENTRY glutRemoveOverlay(void);

GLUTAPI void APIENTRY glutUseLayer(GLenum layer);

GLUTAPI void APIENTRY glutPostOverlayRedisplay(void);

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 11)

GLUTAPI void APIENTRY glutPostWindowOverlayRedisplay(int win);

#endif

GLUTAPI void APIENTRY glutShowOverlay(void);

GLUTAPI void APIENTRY glutHideOverlay(void);

#endif

GLUTAPI int APIENTRY glutCreateMenu(void (GLUTCALLBACK \*func)(int));

#if defined(\_WIN32) && !defined(GLUT\_DISABLE\_ATEXIT\_HACK)

GLUTAPI int APIENTRY \_\_glutCreateMenuWithExit(void (GLUTCALLBACK \*func)(int), void (\_\_cdecl \*exitfunc)(int));

#ifndef GLUT\_BUILDING\_LIB

static int APIENTRY glutCreateMenu\_ATEXIT\_HACK(void (GLUTCALLBACK \*func)(int)) { return \_\_glutCreateMenuWithExit(func, exit); }

#define glutCreateMenu glutCreateMenu\_ATEXIT\_HACK

#endif

#endif

GLUTAPI void APIENTRY glutDestroyMenu(int menu);

GLUTAPI int APIENTRY glutGetMenu(void);

GLUTAPI void APIENTRY glutSetMenu(int menu);

GLUTAPI void APIENTRY glutAddMenuEntry(const char \*label, int value);

GLUTAPI void APIENTRY glutAddSubMenu(const char \*label, int submenu);

GLUTAPI void APIENTRY glutChangeToMenuEntry(int item, const char \*label, int value);

GLUTAPI void APIENTRY glutChangeToSubMenu(int item, const char \*label, int submenu);

GLUTAPI void APIENTRY glutRemoveMenuItem(int item);

GLUTAPI void APIENTRY glutAttachMenu(int button);

GLUTAPI void APIENTRY glutDetachMenu(int button);

GLUTAPI void APIENTRY glutDisplayFunc(void (GLUTCALLBACK \*func)(void));

GLUTAPI void APIENTRY glutReshapeFunc(void (GLUTCALLBACK \*func)(int width, int height));

GLUTAPI void APIENTRY glutKeyboardFunc(void (GLUTCALLBACK \*func)(unsigned char key, int x, int y));

GLUTAPI void APIENTRY glutMouseFunc(void (GLUTCALLBACK \*func)(int button, int state, int x, int y));

GLUTAPI void APIENTRY glutMotionFunc(void (GLUTCALLBACK \*func)(int x, int y));

GLUTAPI void APIENTRY glutPassiveMotionFunc(void (GLUTCALLBACK \*func)(int x, int y));

GLUTAPI void APIENTRY glutEntryFunc(void (GLUTCALLBACK \*func)(int state));

GLUTAPI void APIENTRY glutVisibilityFunc(void (GLUTCALLBACK \*func)(int state));

GLUTAPI void APIENTRY glutIdleFunc(void (GLUTCALLBACK \*func)(void));

GLUTAPI void APIENTRY glutTimerFunc(unsigned int millis, void (GLUTCALLBACK \*func)(int value), int value);

GLUTAPI void APIENTRY glutMenuStateFunc(void (GLUTCALLBACK \*func)(int state));

#if (GLUT\_API\_VERSION >= 2)

GLUTAPI void APIENTRY glutSpecialFunc(void (GLUTCALLBACK \*func)(int key, int x, int y));

GLUTAPI void APIENTRY glutSpaceballMotionFunc(void (GLUTCALLBACK \*func)(int x, int y, int z));

GLUTAPI void APIENTRY glutSpaceballRotateFunc(void (GLUTCALLBACK \*func)(int x, int y, int z));

GLUTAPI void APIENTRY glutSpaceballButtonFunc(void (GLUTCALLBACK \*func)(int button, int state));

GLUTAPI void APIENTRY glutButtonBoxFunc(void (GLUTCALLBACK \*func)(int button, int state));

GLUTAPI void APIENTRY glutDialsFunc(void (GLUTCALLBACK \*func)(int dial, int value));

GLUTAPI void APIENTRY glutTabletMotionFunc(void (GLUTCALLBACK \*func)(int x, int y));

GLUTAPI void APIENTRY glutTabletButtonFunc(void (GLUTCALLBACK \*func)(int button, int state, int x, int y));

#if (GLUT\_API\_VERSION >= 3)

GLUTAPI void APIENTRY glutMenuStatusFunc(void (GLUTCALLBACK \*func)(int status, int x, int y));

GLUTAPI void APIENTRY glutOverlayDisplayFunc(void (GLUTCALLBACK \*func)(void));

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 9)

GLUTAPI void APIENTRY glutWindowStatusFunc(void (GLUTCALLBACK \*func)(int state));

#endif

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 13)

GLUTAPI void APIENTRY glutKeyboardUpFunc(void (GLUTCALLBACK \*func)(unsigned char key, int x, int y));

GLUTAPI void APIENTRY glutSpecialUpFunc(void (GLUTCALLBACK \*func)(int key, int x, int y));

GLUTAPI void APIENTRY glutJoystickFunc(void (GLUTCALLBACK \*func)(unsigned int buttonMask, int x, int y, int z), int pollInterval);

#endif

#endif

#endif

GLUTAPI void APIENTRY glutSetColor(int, GLfloat red, GLfloat green, GLfloat blue);

GLUTAPI GLfloat APIENTRY glutGetColor(int ndx, int component);

GLUTAPI void APIENTRY glutCopyColormap(int win);

GLUTAPI int APIENTRY glutGet(GLenum type);

GLUTAPI int APIENTRY glutDeviceGet(GLenum type);

#if (GLUT\_API\_VERSION >= 2)

GLUTAPI int APIENTRY glutExtensionSupported(const char \*name);

#endif

#if (GLUT\_API\_VERSION >= 3)

GLUTAPI int APIENTRY glutGetModifiers(void);

GLUTAPI int APIENTRY glutLayerGet(GLenum type);

#endif

GLUTAPI void APIENTRY glutBitmapCharacter(void \*font, int character);

GLUTAPI int APIENTRY glutBitmapWidth(void \*font, int character);

GLUTAPI void APIENTRY glutStrokeCharacter(void \*font, int character);

GLUTAPI int APIENTRY glutStrokeWidth(void \*font, int character);

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 9)

GLUTAPI int APIENTRY glutBitmapLength(void \*font, const unsigned char \*string);

GLUTAPI int APIENTRY glutStrokeLength(void \*font, const unsigned char \*string);

#endif

GLUTAPI void APIENTRY glutWireSphere(GLdouble radius, GLint slices, GLint stacks);

GLUTAPI void APIENTRY glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);

GLUTAPI void APIENTRY glutWireCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);

GLUTAPI void APIENTRY glutSolidCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);

GLUTAPI void APIENTRY glutWireCube(GLdouble size);

GLUTAPI void APIENTRY glutSolidCube(GLdouble size);

GLUTAPI void APIENTRY glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);

GLUTAPI void APIENTRY glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);

GLUTAPI void APIENTRY glutWireDodecahedron(void);

GLUTAPI void APIENTRY glutSolidDodecahedron(void);

GLUTAPI void APIENTRY glutWireTeapot(GLdouble size);

GLUTAPI void APIENTRY glutSolidTeapot(GLdouble size);

GLUTAPI void APIENTRY glutWireOctahedron(void);

GLUTAPI void APIENTRY glutSolidOctahedron(void);

GLUTAPI void APIENTRY glutWireTetrahedron(void);

GLUTAPI void APIENTRY glutSolidTetrahedron(void);

GLUTAPI void APIENTRY glutWireIcosahedron(void);

GLUTAPI void APIENTRY glutSolidIcosahedron(void);

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 9)

GLUTAPI int APIENTRY glutVideoResizeGet(GLenum param);

GLUTAPI void APIENTRY glutSetupVideoResizing(void);

GLUTAPI void APIENTRY glutStopVideoResizing(void);

GLUTAPI void APIENTRY glutVideoResize(int x, int y, int width, int height);

GLUTAPI void APIENTRY glutVideoPan(int x, int y, int width, int height);

GLUTAPI void APIENTRY glutReportErrors(void);

#endif

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 13)

#define GLUT\_KEY\_REPEAT\_OFF 0

#define GLUT\_KEY\_REPEAT\_ON 1

#define GLUT\_KEY\_REPEAT\_DEFAULT 2

#define GLUT\_JOYSTICK\_BUTTON\_A 1

#define GLUT\_JOYSTICK\_BUTTON\_B 2

#define GLUT\_JOYSTICK\_BUTTON\_C 4

#define GLUT\_JOYSTICK\_BUTTON\_D 8

GLUTAPI void APIENTRY glutIgnoreKeyRepeat(int ignore);

GLUTAPI void APIENTRY glutSetKeyRepeat(int repeatMode);

GLUTAPI void APIENTRY glutForceJoystickFunc(void);

#define GLUT\_GAME\_MODE\_ACTIVE ((GLenum) 0)

#define GLUT\_GAME\_MODE\_POSSIBLE ((GLenum) 1)

#define GLUT\_GAME\_MODE\_WIDTH ((GLenum) 2)

#define GLUT\_GAME\_MODE\_HEIGHT ((GLenum) 3)

#define GLUT\_GAME\_MODE\_PIXEL\_DEPTH ((GLenum) 4)

#define GLUT\_GAME\_MODE\_REFRESH\_RATE ((GLenum) 5)

#define GLUT\_GAME\_MODE\_DISPLAY\_CHANGED ((GLenum) 6)

GLUTAPI void APIENTRY glutGameModeString(const char \*string);

GLUTAPI int APIENTRY glutEnterGameMode(void);

GLUTAPI void APIENTRY glutLeaveGameMode(void);

GLUTAPI int APIENTRY glutGameModeGet(GLenum mode);

#endif

#ifdef \_\_cplusplus

}

#endif

#ifdef GLUT\_APIENTRY\_DEFINED

# undef GLUT\_APIENTRY\_DEFINED

# undef APIENTRY

#endif

#ifdef GLUT\_WINGDIAPI\_DEFINED

# undef GLUT\_WINGDIAPI\_DEFINED

# undef WINGDIAPI

#endif

#ifdef GLUT\_DEFINED\_\_\_CDECL

# undef GLUT\_DEFINED\_\_\_CDECL

# undef \_\_cdecl

#endif

#ifdef GLUT\_DEFINED\_\_CRTIMP

# undef GLUT\_DEFINED\_\_CRTIMP

# undef \_CRTIMP

#endif

#endif

**Bullet.cpp:**#include "Bullet.h"

**קבצי Source (cpp.):**

Bullet::Bullet()

{

}

void Bullet::BulletShot()

{

MoveModel(Z\_AXIS, -0.01667);

MoveModel(Y\_AXIS, 0.002);

}

void Bullet::BulletComeBack()

{

MoveModel(Z\_AXIS, 6.0012);

MoveModel(Y\_AXIS, -0.72);

}

void Bullet::BulletHitCheck(GameState &gs1, Target &target)

{

if (GetModelCenter().GetCoordinate(X\_AXIS) >= (target.GetPolygon(0).GetPolygonCenter().GetCoordinate(X\_AXIS) - 1.3) && GetModelCenter().GetCoordinate(X\_AXIS) <= (target.GetPolygon(0).GetPolygonCenter().GetCoordinate(X\_AXIS) + 1.3))

{

gs1.SetBullsEye(false);

gs1.SetShotMissed(false);

gs1.SetBulletHitTarget(true);

gs1.SetLevelNum(gs1.GetLevelNum() + 1);

BulletComeBack();

if (GetModelCenter().GetCoordinate(X\_AXIS) >= (target.GetPolygon(0).GetPolygonCenter().GetCoordinate(X\_AXIS) - 0.2184) && GetModelCenter().GetCoordinate(X\_AXIS) <= (target.GetPolygon(0).GetPolygonCenter().GetCoordinate(X\_AXIS) + 0.2184))

{

gs1.SetBullsEye(true);

}

}

else

{

gs1.SetBullsEye(false);

gs1.SetShotMissed(true);

gs1.SetBulletHitTarget(false);

BulletComeBack();

gs1.SetLives(gs1.GetLives() - 1);

}

}

**CBall.cpp:**#include <math.h>

#include "CBall.h"

const double pi = (double)3.1415926536;

CBall::CBall(void)

{

int num\_of\_fi\_values = 10;

int num\_of\_theta\_values = 30;

SetBallNumOfElements(num\_of\_fi\_values, num\_of\_theta\_values);

}

CBall::CBall(int num\_of\_fi\_values, int num\_of\_theta\_values)

{

SetBallNumOfElements(num\_of\_fi\_values, num\_of\_theta\_values);

}

void CBall::SetBallNumOfElements(int num\_of\_fi\_values, int num\_of\_theta\_values)

{

\_num\_of\_fi\_values = num\_of\_fi\_values;

\_num\_of\_theta\_values = num\_of\_theta\_values;

bool divide\_by\_area = false;

int total\_num\_of\_polygons = num\_of\_fi\_values\*num\_of\_theta\_values;

SetNumberOfPolygons(total\_num\_of\_polygons);

double x, y, z;

double delta\_fi = pi/num\_of\_fi\_values;

double delta\_theta = 2\*pi/num\_of\_theta\_values;

double current\_fi;

double next\_fi = 0;

for(int i = 0; i < num\_of\_fi\_values; i++)

{

double small\_fi;

double big\_fi;

if (divide\_by\_area == false)

{

small\_fi = i\*delta\_fi;

big\_fi = (i+1)\*delta\_fi;

}

else

{

current\_fi = next\_fi;

next\_fi = acos(cos(current\_fi)-2.0/num\_of\_fi\_values);

small\_fi = current\_fi;

big\_fi = next\_fi;

}

for(int j = 0;j < num\_of\_theta\_values;j++)

{

int polygon\_num = i\*num\_of\_theta\_values+j;

CPolygon current\_polygon(4);

double small\_theta = j\*delta\_theta;

double big\_theta = (j+1)\*delta\_theta;

x = sin(small\_fi)\*cos(small\_theta);

y = sin(small\_fi)\*sin(small\_theta);

z = cos(small\_fi);

current\_polygon.SetOnePointCoordinates(0, x, y, z);

x = sin(big\_fi)\*cos(small\_theta);

y = sin(big\_fi)\*sin(small\_theta);

z = cos(big\_fi);

current\_polygon.SetOnePointCoordinates(1, x, y, z);

x = sin(big\_fi)\*cos(big\_theta);

y = sin(big\_fi)\*sin(big\_theta);

z = cos(big\_fi);

current\_polygon.SetOnePointCoordinates(2, x, y, z);

x = sin(small\_fi)\*cos(big\_theta);

y = sin(small\_fi)\*sin(big\_theta);

z = cos(small\_fi);

current\_polygon.SetOnePointCoordinates(3, x, y, z);

SetOnePolygon(polygon\_num, current\_polygon);

}

}

SetBallColors(0,1,0,1,0,1);

}

void CBall::SetBallColors(double red\_component\_min, double red\_component\_max, double green\_component\_min, double green\_component\_max, double blue\_component\_min, double blue\_component\_max)

{

double red\_component\_diff = red\_component\_max - red\_component\_min;

double green\_component\_diff = green\_component\_max - green\_component\_min;

double blue\_component\_diff = blue\_component\_max - blue\_component\_min;

for(int i = 0; i < \_num\_of\_fi\_values; i++)

for(int j = 0;j < \_num\_of\_theta\_values;j++)

{

int polygon\_num = i\*\_num\_of\_theta\_values+j;

double red = GetNewColorComponent(i,j,1,1)\*red\_component\_diff + red\_component\_min;

double green = GetNewColorComponent(i,j,1,2)\*green\_component\_diff + green\_component\_min;

double blue = GetNewColorComponent(i,j,3,2)\*blue\_component\_diff + blue\_component\_min;

SetOnePolygonColor(polygon\_num, red, green, blue);

}

}

double CBall::GetNewColorComponent(int fi\_index, int theta\_index, int fi\_weight, int theta\_weight)

{

double fi\_val = fi\_index/(1.0\*\_num\_of\_fi\_values);

double theta\_val = 0.5\*(1+sin(theta\_index/(1.0\*\_num\_of\_theta\_values)\*2\*pi));

double new\_color = (fi\_val\*fi\_weight + theta\_val\*theta\_weight)/(fi\_weight + theta\_weight);

return new\_color;

}

**CCircle.cpp:**#include "CCircle.h"

#include <math.h>

const float PI = (float)3.1415926536;

CCircle::CCircle(void)

{

SetNumberOfPoints(30);

}

CCircle::CCircle(int num\_of\_points)

{

SetNumberOfPoints(num\_of\_points);

}

void CCircle::SetNumberOfPoints(int num\_of\_points)

{

CPolygon::SetNumberOfPoints(num\_of\_points);

double delta\_angle = 2 \* PI / num\_of\_points;

for (int i = 0; i < num\_of\_points; i++)

{

double angle = i \* delta\_angle;

double x = cos(angle);

double y = sin(angle);

SetOnePointCoordinates(i, x, y, 0);

}

SetPolygonColor(0, 0, 0);

}  
**CGameHandler.cpp:**#include <iostream>

#include <math.h>

#include "glut.h"

#include "CGameHandler.h"

#include "stdio.h"

#include "Gun.h"

#include "Target.h"

#include "Bullet.h"

#include "GameState.h"

GameState gs1;

int BulletShotTimer = 360;

char Level[128];

char HP[128];

char stages[128];

Gun gun1;

Bullet bullet;

Target target;

extern void renderBitmapString(float x, float y, void \*font, char \*string);

extern void renderVerticalBitmapString(float x, float y, int bitmapHeight, void \*font, char \*string);

void CGameHandler::StartOfGame()

{

font = def\_font = (int)GLUT\_BITMAP\_HELVETICA\_18;

ChangeBackgroundColors(0, 0, 0);

gun1.MoveModel(3.25, -3, -10);

gun1.MoveModel(Z\_AXIS, 2.5);

gun1.MoveModel(Y\_AXIS, 1);

gun1.RotateSelfModel(X\_AXIS, 3);

bullet.ScaleModel(0.125, 0.125, 0.125);

bullet.MoveModel(gun1.GetPolygon(22).GetPolygonCenter());

bullet.SetBallColors(0, 0.5, 0, 0.3, 0, 0);

target.GetPolygon(0).MovePolygonCenter(-0.142, 0.7, -15);

}

void CGameHandler::DrawScene()

{

gun1.DrawModel();

bullet.DrawModel();

target.DrawModel();

}

void CGameHandler::LogicPart()

{

if (gs1.GetGameStarted() == true && gs1.GetGameOver() == false && gs1.GetCongratz() == false)

{

if (gs1.GetGameStarted() == true)

{

target.TargetMovement(gs1);

}

if (gs1.GetBBS() == true && BulletShotTimer > 0)

{

gs1.SetFirstShotBeenShot(true);

bullet.BulletShot();

BulletShotTimer -= 1;

}

if (gs1.GetBBS() == true && BulletShotTimer == 0)

{

bullet.BulletHitCheck(gs1, target);

BulletShotTimer = 360;

gs1.SetBBS(false);

}

}

if (gs1.GetLives() == 0)

{

gs1.Gameover();

}

if (gs1.GetStage() == 2 && gs1.GetLevelNum() > 10)

{

gs1.SetStageUp(false);

}

if ((gs1.GetStage() == 1 || gs1.GetStage() == 2) && gs1.GetLevelNum() > 10)

{

gs1.StageChange();

}

if (gs1.GetStage() == 3 && gs1.GetLevelNum() > 10)

{

gs1.GameFinished();

}

if (gs1.GetStage() == 2 && gs1.GetStageUp() == false)

{

gs1.Stage2LivePlus1();

}

if (gs1.GetStage() == 3 && gs1.GetStageUp() == false)

{

gs1.Stage3LivePlus3();

}

}

void CGameHandler::WriteText()

{

font = (int)GLUT\_BITMAP\_TIMES\_ROMAN\_24;

glColor3f(1, 1, 0);

int x\_pos = glutGet(GLUT\_SCREEN\_WIDTH) / 2 - 100;

int y\_pos = 30;

renderBitmapString(x\_pos, y\_pos, (void \*)font, "Target Shooting");

glColor3f(0, 1, 0);

if (gs1.GetCongratz() == false)

{

sprintf(Level, "Level: %d", gs1.GetLevelNum());

if (gs1.GetLevelNum() >= 1)

renderBitmapString(x\_pos, y\_pos + 60, (void \*)font, Level);

glColor3f(0, 0, 1);

if (gs1.GetBulletHitTarget() == true && gs1.GetFirstShotBeenShot() == true && gs1.GetBullsEye() == false)

renderBitmapString(x\_pos, y\_pos + 90, (void \*)font, "Nice Shot!");

if (gs1.GetShotMissed() == true && gs1.GetFirstShotBeenShot() == true && gs1.GetBullsEye() == false)

renderBitmapString(x\_pos, y\_pos + 90, (void \*)font, "Miss!");

if (gs1.GetBullsEye() == true && gs1.GetFirstShotBeenShot() == true)

renderBitmapString(x\_pos, y\_pos + 90, (void \*)font, "BULLS EYE!");

glColor3f(1, 0, 0);

sprintf(HP, "Lives: %d", gs1.GetLives());

if (gs1.GetLevelNum() >= 1)

renderBitmapString(x\_pos - 850, y\_pos, (void \*)font, HP);

glColor3f(1, 0, 1);

sprintf(stages, "Stage: %d", gs1.GetStage());

if (gs1.GetLevelNum() >= 1)

renderBitmapString(x\_pos, y\_pos + 30, (void \*)font, stages);

if (gs1.GetStage() == 2 && gs1.GetLevelNum() < 3)

renderBitmapString(x\_pos + 90, y\_pos + 30, (void \*)font, "The Target Will Now Keep Moving When You Shoot !");

if (gs1.GetStage() == 3 && gs1.GetLevelNum() < 3)

{

renderBitmapString(x\_pos + 90, y\_pos + 30, (void \*)font, "The Target Will Now Keep Moving When You Shoot,");

renderBitmapString(x\_pos + 90, y\_pos + 60, (void \*)font, "And Change Direction When You Shoot !");

}

glColor3f(1, 0, 0);

if (gs1.GetLives() == 0)

{

renderBitmapString(x\_pos + 425, y\_pos + 200, (void \*)font, "Game Over!");

renderBitmapString(x\_pos + 375, y\_pos + 230, (void \*)font, "Press 'R' To Start Again");

}

glColor3f(1, 1, 1);

if (gs1.GetGameStarted() == false)

{

renderBitmapString(x\_pos + 425, y\_pos + 200, (void \*)font, "Press Space To Start");

}

if (gs1.GetGameStarted() == true && gs1.GetLives() != 0)

{

renderBitmapString(x\_pos + 425, y\_pos + 100, (void \*)font, "Press Space to Shoot");

renderBitmapString(x\_pos + 425, y\_pos + 130, (void \*)font, "<-- Button moves the gun left");

renderBitmapString(x\_pos + 425, y\_pos + 160, (void \*)font, "--> Button moves the gun right");

glColor3f(1, 0, 0);

renderBitmapString(x\_pos + 425, y\_pos + 190, (void \*)font, "Press 'R' To Restart");

}

}

if (gs1.GetCongratz() == true)

{

glColor3f(1, 1, 1);

renderBitmapString(x\_pos + 425, y\_pos + 100, (void \*)font, "Congratulations !");

renderBitmapString(x\_pos + 425, y\_pos + 130, (void \*)font, "You Finished The Game !");

renderBitmapString(x\_pos + 425, y\_pos + 160, (void \*)font, "Well Done !");

glColor3f(1, 0, 0);

renderBitmapString(x\_pos + 425, y\_pos + 190, (void \*)font, "Press 'R' To Restart");

}

if (gs1.GetStageUp() == true && gs1.GetLevelNum() == 1)

{

if (gs1.GetStage() == 2)

{

renderBitmapString(x\_pos - 600, y\_pos + 160, (void \*)font, "One Life Was Added");

renderBitmapString(x\_pos - 615, y\_pos + 190, (void \*)font, "For Getting To Stage 2 !");

}

if (gs1.GetStage() == 3)

{

renderBitmapString(x\_pos - 600, y\_pos + 160, (void \*)font, "Three Lifes Were Added");

renderBitmapString(x\_pos - 615, y\_pos + 190, (void \*)font, "For Getting To Stage 3 !");

}

}

}

void CGameHandler::KeyPressed(char key)

{

switch (key)

{

case ' ':

{

gs1.ShootingHandler();

break;

}

case 'r':

{

gs1.RESET();

target.MoveModelCenter(-0.142, 0.7, -15);

break;

}

}

}

void CGameHandler::SpecialKeyPressed(char key)

{

switch (key)

{

case GLUT\_KEY\_LEFT:

if (gun1.GetModelCenter().GetCoordinate(X\_AXIS) > -4 && gs1.GetBBS() == false)

{

gun1.MoveModel(X\_AXIS, -1 \* gs1.GetGunMove());

bullet.MoveModel(X\_AXIS, -1 \* gs1.GetGunMove());

}

break;

case GLUT\_KEY\_RIGHT:

if (gun1.GetModelCenter().GetCoordinate(X\_AXIS) < 4 && gs1.GetBBS() == false)

{

gun1.MoveModel(X\_AXIS, gs1.GetGunMove());

bullet.MoveModel(X\_AXIS, gs1.GetGunMove());

}

break;

case GLUT\_KEY\_UP:

break;

case GLUT\_KEY\_DOWN:

default:

break;

}

}

float CGameHandler::GetBeckgroundColorR()

{

return BGColor\_R;

}

float CGameHandler::GetBeckgroundColorG()

{

return BGColor\_G;

}

float CGameHandler::GetBeckgroundColorB()

{

return BGColor\_B;

}

void CGameHandler::ChangeBackgroundColors(float R, float G, float B)

{

BGColor\_R = R;

BGColor\_G = G;

BGColor\_B = B;

}

**CModel.cpp:**

#include "CModel.h"

#pragma region CONSTRUCTORs & DESTRUCTOR

CModel::CModel()

{

\_num\_of\_polygons = 2;

\_polygons\_array = new CPolygon[\_num\_of\_polygons];

}

CModel::CModel(int num\_of\_polygons)

{

\_polygons\_array = new CPolygon[\_num\_of\_polygons];

}

CModel::CModel(const CModel& model)

{

\_num\_of\_polygons = model.\_num\_of\_polygons;

\_polygons\_array = new CPolygon[\_num\_of\_polygons];

for (int i = 0; i < \_num\_of\_polygons; i++)

\_polygons\_array[i].DuplicatePolygon(model.\_polygons\_array[i]);

}

CModel::~CModel()

{

delete[] \_polygons\_array;

}

#pragma endregion

#pragma region Setting model properties

void CModel::SetNumberOfPolygons(int num\_of\_polygons)

{

delete[] \_polygons\_array;

\_num\_of\_polygons = num\_of\_polygons;

\_polygons\_array = new CPolygon[\_num\_of\_polygons];

}

void CModel::SetModelColor(Color color\_component, double color\_value)

{

for (int i = 0; i < \_num\_of\_polygons; i++)

\_polygons\_array[i].SetPolygonColor(color\_component, color\_value);

}

void CModel::SetModelColor(double red\_component, double green\_component, double blue\_component)

{

for (int i = 0; i < \_num\_of\_polygons; i++)

{

\_polygons\_array[i].SetPolygonColor(red\_component, green\_component, blue\_component);

}

}

#pragma endregion

#pragma region Setting one polygon properties

void CModel::SetOnePolygon(int polygon\_index, CPolygon polygon)

{

\_polygons\_array[polygon\_index].DuplicatePolygon(polygon);

}

void CModel::SetOnePolygonColor(int polygon\_index, Color color\_component, double color\_value)

{

\_polygons\_array[polygon\_index].SetPolygonColor(color\_component, color\_value);

}

void CModel::SetOnePolygonColor(int polygon\_index, double red\_component, double green\_component, double blue\_component)

{

\_polygons\_array[polygon\_index].SetPolygonColor(red\_component, green\_component, blue\_component);

}

#pragma endregion

#pragma region Move model functions

void CModel::MoveModelTo(double x\_Coordinate, double y\_Coordinate, double z\_Coordinate)

{

for (int i = 0; i < \_num\_of\_polygons; i++)

{

\_polygons\_array[i].MovePolygonTo(x\_Coordinate,y\_Coordinate,z\_Coordinate);

}

}

void CModel::MoveModel(Axis axis, double move\_value)

{

for (int i = 0; i < \_num\_of\_polygons; i++)

{

\_polygons\_array[i].MovePolygon(axis, move\_value);

}

}

void CModel::MoveModel(double x\_move, double y\_move, double z\_move)

{

for (int i = 0; i < \_num\_of\_polygons; i++)

{

\_polygons\_array[i].MovePolygon(x\_move, y\_move, z\_move);

}

}

void CModel::MoveModel(CPoint point\_move)

{

for (int i = 0; i < \_num\_of\_polygons; i++)

{

\_polygons\_array[i].MovePolygon(point\_move);

}

}

void CModel::MoveModelCenter(Axis axis, double coordinate\_value)

{

CPoint p = GetModelCenter();

p.ScalePoint(-1, -1, -1);

MoveModel(p);

MoveModel(axis, coordinate\_value);

}

void CModel::MoveModelCenter(double x\_coordinate, double y\_coordinate, double z\_coordinate)

{

CPoint p = GetModelCenter();

p.ScalePoint(-1, -1, -1);

MoveModel(p);

MoveModel(x\_coordinate, y\_coordinate, z\_coordinate);

}

void CModel::MoveModelCenter(CPoint point\_coordinate)

{

CPoint p = GetModelCenter();

p.ScalePoint(-1, -1, -1);

MoveModel(p);

MoveModel(point\_coordinate);

}

#pragma endregion

#pragma region Scale model functions

void CModel::ScaleModel(Axis axis, double scale\_ratio)

{

for (int i = 0; i < \_num\_of\_polygons; i++)

{

\_polygons\_array[i].ScalePolygon(axis, scale\_ratio);

}

}

void CModel::ScaleModel(double x\_ratio, double y\_ratio, double z\_ratio)

{

for (int i = 0; i < \_num\_of\_polygons; i++)

{

\_polygons\_array[i].ScalePolygon(x\_ratio, y\_ratio, z\_ratio);

}

}

void CModel::ScaleSelfModel(Axis axis, double scale\_ratio)

{

for (int i = 0; i < \_num\_of\_polygons; i++)

{

\_polygons\_array[i].ScaleSelfPolygon(axis, scale\_ratio);

}

}

void CModel::ScaleSelfModel(double x\_ratio, double y\_ratio, double z\_ratio)

{

for (int i = 0; i < \_num\_of\_polygons; i++)

{

\_polygons\_array[i].ScaleSelfPolygon(x\_ratio, y\_ratio, z\_ratio);

}

}

#pragma endregion

#pragma region Rotate model functions

void CModel::RotateModel(Axis axis, double rotate\_value)

{

for (int i = 0; i < \_num\_of\_polygons; i++)

{

\_polygons\_array[i].RotatePolygon(axis, rotate\_value);

}

}

void CModel::RotateSelfModel(Axis axis, double rotate\_value)

{

CPoint model\_center = GetModelCenter();

model\_center.ScalePoint(-1, -1, -1);

MoveModel(model\_center);

model\_center.ScalePoint(-1, -1, -1);

RotateModel(axis, rotate\_value);

MoveModel(model\_center);

}

#pragma endregion

#pragma region Draw model function

void CModel::DrawModel()

{

for (int i = 0; i < \_num\_of\_polygons; i++)

\_polygons\_array[i].DrawPolygon();

}

#pragma endregion

#pragma region Get functions

CPolygon CModel::GetPolygon(int polygon\_index)

{

return \_polygons\_array[polygon\_index];

}

CPoint CModel::GetModelCenter()

{

CPoint sum1(0, 0, 0);

for (int i = 0; i < \_num\_of\_polygons; i++)

{

sum1.MovePoint(\_polygons\_array[i].GetPolygonCenter());

}

double scaleRatio = 1.0 / \_num\_of\_polygons;

sum1.ScalePoint(scaleRatio, scaleRatio, scaleRatio);

return sum1;

}

int CModel::GetNumberOfPolygons()

{

return \_num\_of\_polygons;

}

#pragma endregion

**CPolygon:**#include < stdlib.h>

#include "glut.h"

#include <math.h>

#include "CPolygon.h"

const float PI = (float)3.1415926536;

#pragma region CONSTRUCTORs & DESTRUCTOR

CPolygon::CPolygon()

{

\_num\_of\_points = 2;

\_points\_array = new CPoint[\_num\_of\_points];

}

CPolygon::CPolygon(int num\_of\_points)

{

\_num\_of\_points = num\_of\_points;

\_points\_array = new CPoint[\_num\_of\_points];

}

CPolygon::CPolygon(const CPolygon& polygon)

{

\_num\_of\_points = polygon.\_num\_of\_points;

\_points\_array = new CPoint[\_num\_of\_points];

for (int i = 0; i < \_num\_of\_points; i++)

\_points\_array[i] = polygon.\_points\_array[i].ClonePoint();

}

CPolygon::~CPolygon()

{

delete[] \_points\_array;

}

#pragma endregion

#pragma region Setting CPolygon properties

void CPolygon::SetNumberOfPoints(int num\_of\_points)

{

delete[] \_points\_array;

\_num\_of\_points = num\_of\_points;

\_points\_array = new CPoint[\_num\_of\_points];

}

void CPolygon::SetPolygonColor(Color color\_component, double color\_value)

{

for (int i = 0; i < \_num\_of\_points; i++)

\_points\_array[i].SetPointColor(color\_component, color\_value);

}

void CPolygon::SetPolygonColor(double red\_component, double green\_component, double blue\_component)

{

for (int i = 0; i < \_num\_of\_points; i++)

{

\_points\_array[i].SetPointColor(red\_component, green\_component, blue\_component);

}

}

void CPolygon::DuplicatePolygon(CPolygon polygon)

{

SetNumberOfPoints(polygon.GetNumberOfPoints());

for (int i = 0; i < \_num\_of\_points; i++)

SetOnePoint(i, polygon.GetPoint(i));

}

#pragma endregion

#pragma region Setting one point properties

void CPolygon::SetOnePoint(int point\_index, CPoint point)

{

\_points\_array[point\_index] = point.ClonePoint();

}

void CPolygon::SetOnePointCoordinates(int point\_index, double x\_coordinate, double y\_coordinate, double z\_coordinate)

{

\_points\_array[point\_index].SetPointCoordinates(x\_coordinate, y\_coordinate, z\_coordinate);

}

void CPolygon::SetOnePointColor(int point\_index, Color color\_component, double color\_value)

{

\_points\_array[point\_index].SetPointColor(color\_component, color\_value);

}

void CPolygon::SetOnePointColor(int point\_index, double red\_component, double green\_component, double blue\_component)

{

\_points\_array[point\_index].SetPointColor(red\_component, green\_component, blue\_component);

}

#pragma endregion

#pragma region Move polygon functions

void CPolygon::MovePolygonTo(double x\_Coordinate, double y\_Coordinate, double z\_Coordinate)

{

for (int i = 0; i < \_num\_of\_points; i++)

\_points\_array[i].MovePointTo(x\_Coordinate,y\_Coordinate,z\_Coordinate);

}

void CPolygon::MovePolygon(Axis axis, double move\_value)

{

for (int i = 0; i < \_num\_of\_points; i++)

\_points\_array[i].MovePoint(axis, move\_value);

}

void CPolygon::MovePolygon(double x\_move, double y\_move, double z\_move)

{

for (int i = 0; i < \_num\_of\_points; i++)

{

\_points\_array[i].MovePoint(x\_move, y\_move, z\_move);

}

}

void CPolygon::MovePolygon(CPoint point\_move)

{

for (int i = 0; i < \_num\_of\_points; i++)

{

\_points\_array[i].MovePoint(point\_move);

}

}

void CPolygon::MovePolygonCenter(Axis axis, double coordinate\_value)

{

CPoint p = GetPolygonCenter();

MovePolygon(axis, coordinate\_value - p.GetCoordinate(axis));

}

void CPolygon::MovePolygonCenter(double x\_coordinate, double y\_coordinate, double z\_coordinate)

{

CPoint p1 = GetPolygonCenter();

MovePolygon(x\_coordinate - p1.GetCoordinate(X\_AXIS), y\_coordinate - p1.GetCoordinate(Y\_AXIS), z\_coordinate - p1.GetCoordinate(Z\_AXIS));

}

void CPolygon::MovePolygonCenter(CPoint point\_coordinate)

{

CPoint p = GetPolygonCenter();

p.ScalePoint(-1, -1, -1);

MovePolygon(p);

MovePolygon(point\_coordinate);

}

#pragma endregion

#pragma region Scale polygon functions

void CPolygon::ScalePolygon(Axis axis, double scale\_ratio)

{

for (int i = 0; i < \_num\_of\_points; i++)

\_points\_array[i].ScalePoint(axis, scale\_ratio);

}

void CPolygon::ScalePolygon(double x\_ratio, double y\_ratio, double z\_ratio)

{

for (int i = 0; i < \_num\_of\_points; i++)

{

\_points\_array[i].ScalePoint(x\_ratio, y\_ratio, z\_ratio);

}

}

void CPolygon::ScaleSelfPolygon(Axis axis, double scale\_ratio)

{

CPoint polygon\_center = GetPolygonCenter();

polygon\_center.ScalePoint(-1, -1, -1);

MovePolygon(polygon\_center);

polygon\_center.ScalePoint(-1, -1, -1);

ScalePolygon(axis, scale\_ratio);

MovePolygon(polygon\_center);

}

void CPolygon::ScaleSelfPolygon(double x\_ratio, double y\_ratio, double z\_ratio)

{

CPoint polygon\_center = GetPolygonCenter();

polygon\_center.ScalePoint(-1, -1, -1);

MovePolygon(polygon\_center);

polygon\_center.ScalePoint(-1, -1, -1);

ScalePolygon(x\_ratio,y\_ratio,z\_ratio);

MovePolygon(polygon\_center);

}

#pragma endregion

#pragma region Rotate polygon functions

void CPolygon::RotatePolygon(Axis axis, double rotate\_value)

{

for (int i = 0; i < \_num\_of\_points; i++)

{

\_points\_array[i].RotatePoint(axis, rotate\_value);

}

}

void CPolygon::RotateSelfPolygon(Axis axis, double rotate\_value)

{

CPoint polygon\_center = GetPolygonCenter();

polygon\_center.ScalePoint(-1, -1, -1);

MovePolygon(polygon\_center);

polygon\_center.ScalePoint(-1, -1, -1);

RotatePolygon(axis, rotate\_value);

MovePolygon(polygon\_center);

}

#pragma endregion

#pragma region Draw polygon function

void CPolygon::DrawPolygon()

{

glPolygonMode(GL\_FRONT, GL\_FILL);

glBegin(GL\_POLYGON);

for(int i = 0; i < \_num\_of\_points; i++)

\_points\_array[i].DrawPoint();

glEnd();

}

#pragma endregion

#pragma region Get functions

CPoint CPolygon::GetPoint(int point\_index)

{

return \_points\_array[point\_index];

}

CPoint CPolygon::GetPolygonCenter()

{

CPoint sum(0, 0, 0);

for (int i = 0; i < \_num\_of\_points; i++)

sum.MovePoint(\_points\_array[i]);

double scale\_ratio = 1.0/\_num\_of\_points;

sum.ScalePoint(scale\_ratio, scale\_ratio, scale\_ratio);

return sum;

}

int CPolygon::GetNumberOfPoints()

{

return \_num\_of\_points;

}

#pragma endregion

**CMyOpenGLInit.cpp:**#include "CGameHandler.h"

#include "CMyOpenGLInit.h"

#include "glut.h"

CGameHandler MyGame;

void special(int key, int x, int y);

typedef unsigned char ubyte;

void specialkeys( int key, int x, int y );

void keyboard(unsigned char key, int x, int y);

void setOrthographicProjection()

{

glMatrixMode(GL\_PROJECTION);

glPushMatrix();

glLoadIdentity();

gluOrtho2D(0, glutGet(GLUT\_SCREEN\_WIDTH), 0, glutGet(GLUT\_SCREEN\_HEIGHT));

glScalef(1, -1, 1);

glTranslatef(0, -glutGet(GLUT\_SCREEN\_HEIGHT), 0);

glMatrixMode(GL\_MODELVIEW);

}

void resetPerspectiveProjection()

{

glMatrixMode(GL\_PROJECTION);

glPopMatrix();

glMatrixMode(GL\_MODELVIEW);

}

void renderBitmapString(float x, float y, void \*font, char \*string)

{

char \*c;

glRasterPos2f(x, y);

for (c = string; \*c != '\0'; c++)

{

glutBitmapCharacter(font, \*c);

}

}

void renderSpacedBitmapString(float x, float y, int spacing, void \*font, char \*string)

{

char \*c;

int x1 = x;

for (c = string; \*c != '\0'; c++)

{

glRasterPos2f(x1, y);

glutBitmapCharacter(font, \*c);

x1 = x1 + glutBitmapWidth(font, \*c) + spacing;

}

}

void renderVerticalBitmapString(float x, float y, int bitmapHeight, void \*font, char \*string)

{

char \*c;

int i;

for (c = string, i = 0; \*c != '\0'; i++, c++)

{

glRasterPos2f(x, y+bitmapHeight\*i);

glutBitmapCharacter(font, \*c);

}

}

void MyDraw()

{

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT );

glPushMatrix();

glutPostRedisplay();

glClearColor(MyGame.GetBeckgroundColorR(), MyGame.GetBeckgroundColorG(), MyGame.GetBeckgroundColorB(), 0);

MyGame.DrawScene();

MyGame.LogicPart();

glPushMatrix();

glColor3f(0.0f, 0.0f, 1.0f);

setOrthographicProjection();

glLoadIdentity();

MyGame.WriteText();

resetPerspectiveProjection();

glPopMatrix();

glPopMatrix();

glutSwapBuffers();

}

void reshape( GLsizei width, GLsizei height )

{

GLdouble aspect;

glViewport( 0, 0, width, height );

aspect = (GLdouble) width / (GLdouble) height;

glMatrixMode( GL\_PROJECTION );

glLoadIdentity();

gluPerspective( 45.0, aspect, 1.0, 300.0 );

glMatrixMode( GL\_MODELVIEW );

}

CMyOpenGLInit::CMyOpenGLInit(int argc, char\* argv[])

{

int x, y;

MyGame.StartOfGame();

glutInit(&argc, argv);

x = glutGet(GLUT\_SCREEN\_WIDTH);

y = glutGet(GLUT\_SCREEN\_HEIGHT);

glutInitWindowPosition(0, 0);

glutInitWindowSize(x, y);

glutInitDisplayMode( GLUT\_RGBA | GLUT\_DEPTH | GLUT\_DOUBLE );

glutCreateWindow("Computer Graphics Project");

InitParameters();

glutKeyboardFunc( keyboard );

glutSpecialFunc( special );

glutReshapeFunc( reshape );

glutDisplayFunc (MyDraw);

glClearColor(MyGame.GetBeckgroundColorR(), MyGame.GetBeckgroundColorG(), MyGame.GetBeckgroundColorB(), 0);

glutMainLoop();

}

void CMyOpenGLInit::InitParameters()

{

glEnable( GL\_DEPTH\_TEST );

glEnable( GL\_CULL\_FACE );

glEnable( GL\_LIGHT0 );

glEnable( GL\_LIGHTING );

glEnable( GL\_NORMALIZE );

glColorMaterial( GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE );

glEnable( GL\_COLOR\_MATERIAL );

}

void keyboard(unsigned char key, int x, int y)

{

MyGame.KeyPressed(key);

glutPostRedisplay();

}

void special(int key, int x, int y)

{

MyGame.SpecialKeyPressed(key);

glutPostRedisplay();

}

**CPoint:**

#include <math.h>

#include "CPoint.h"

#include "glut.h"

const float PI = (float)3.1415926536;

#pragma region Constructors

CPoint::CPoint()

{

\_x\_coordinate = 0;

\_y\_coordinate = 0;

\_z\_coordinate = 0;

\_red\_component = 0;

\_green\_component = 0;

\_blue\_component = 0;

}

CPoint::CPoint(double x\_coordinate, double y\_coordinate, double z\_coordinate)

{

\_x\_coordinate = x\_coordinate;

\_y\_coordinate = y\_coordinate;

\_z\_coordinate = z\_coordinate;

\_red\_component = 0;

\_green\_component = 0;

\_blue\_component = 0;

}

CPoint::CPoint(double x\_coordinate, double y\_coordinate, double z\_coordinate, double red\_component, double green\_component, double blue\_component)

{

\_x\_coordinate = x\_coordinate;

\_y\_coordinate = y\_coordinate;

\_z\_coordinate = z\_coordinate;

\_red\_component = red\_component;

\_green\_component = green\_component;

\_blue\_component = blue\_component;

}

#pragma endregion

#pragma region Move point functions

void CPoint::MovePointTo(double x\_Coordinate, double y\_Coordinate, double z\_Coordinate)

{

\_x\_coordinate = x\_Coordinate;

\_y\_coordinate = y\_Coordinate;

\_z\_coordinate = z\_Coordinate;

}

void CPoint::MovePoint(Axis axis, double move\_value)

{

switch (axis)

{

case X\_AXIS:

\_x\_coordinate += move\_value;

break;

case Y\_AXIS:

\_y\_coordinate += move\_value;

break;

case Z\_AXIS:

\_z\_coordinate += move\_value;

break;

}

}

void CPoint::MovePointX(double x\_move)

{

\_x\_coordinate += x\_move;

}

void CPoint::MovePointY(double y\_move)

{

\_y\_coordinate += y\_move;

}

void CPoint::MovePointZ(double z\_move)

{

\_z\_coordinate += z\_move;

}

void CPoint::MovePoint(double x\_move, double y\_move, double z\_move)

{

\_x\_coordinate += x\_move;

\_y\_coordinate += y\_move;

\_z\_coordinate += z\_move;

}

void CPoint::MovePoint(CPoint point\_move)

{

MovePointX(point\_move.GetX());

MovePointY(point\_move.GetY());

MovePointZ(point\_move.GetZ());

}

#pragma endregion

#pragma region Set point location functions

void CPoint::SetPointCoordinate(Axis axis, double coordinate\_value)

{

switch (axis)

{

case X\_AXIS:

SetPointX(coordinate\_value);

break;

case Y\_AXIS:

SetPointY(coordinate\_value);

break;

case Z\_AXIS:

SetPointZ(coordinate\_value);

break;

}

}

void CPoint::SetPointX(double x\_coordinate)

{

\_x\_coordinate = x\_coordinate;

}

void CPoint::SetPointY(double y\_coordinate)

{

\_y\_coordinate = y\_coordinate;

}

void CPoint::SetPointZ(double z\_coordinate)

{

\_z\_coordinate = z\_coordinate;

}

void CPoint::SetPointCoordinates(double x\_coordinate, double y\_coordinate, double z\_coordinate)

{

SetPointX(x\_coordinate);

SetPointY(y\_coordinate);

SetPointZ(z\_coordinate);

}

#pragma endregion

#pragma region Scale point functions

void CPoint::ScalePoint(Axis axis, double scale\_ratio)

{

switch (axis)

{

case X\_AXIS:

\_x\_coordinate \*= scale\_ratio;

break;

case Y\_AXIS:

\_y\_coordinate \*= scale\_ratio;

break;

case Z\_AXIS:

\_z\_coordinate \*= scale\_ratio;

break;

}

}

void CPoint::ScalePoint(double x\_ratio, double y\_ratio, double z\_ratio)

{

\_x\_coordinate \*= x\_ratio;

\_y\_coordinate \*= y\_ratio;

\_z\_coordinate \*= z\_ratio;

}

#pragma endregion

#pragma region Rotate point functions

void CPoint::RotatePoint(Axis axis, double rotate\_value)

{

switch (axis)

{

case X\_AXIS:

RotatePointX(rotate\_value);

break;

case Y\_AXIS:

RotatePointY(rotate\_value);

break;

case Z\_AXIS:

RotatePointZ(rotate\_value);

break;

}

}

void CPoint::RotatePointX(double x\_rotate)

{

double angle\_in\_radians = x\_rotate \* PI / 180;

double original\_y\_coordinate = \_y\_coordinate;

double cos\_value = cos(angle\_in\_radians);

double sin\_value = sin(angle\_in\_radians);

\_y\_coordinate = original\_y\_coordinate \* cos\_value - \_z\_coordinate \* sin\_value;

\_z\_coordinate = original\_y\_coordinate \* sin\_value + \_z\_coordinate \* cos\_value;

}

void CPoint::RotatePointY(double y\_rotate)

{

double angle\_in\_radians = y\_rotate \* PI / 180;

double original\_x\_coordinate = \_x\_coordinate;

double cos\_value = cos(angle\_in\_radians);

double sin\_value = sin(angle\_in\_radians);

\_x\_coordinate = \_z\_coordinate \* sin\_value + original\_x\_coordinate \* cos\_value;

\_z\_coordinate = \_z\_coordinate \* cos\_value - original\_x\_coordinate \* sin\_value;

}

void CPoint::RotatePointZ(double z\_rotate)

{

double angle\_in\_radians = z\_rotate \* PI / 180;

double original\_x\_coordinate = \_x\_coordinate;

double cos\_value = cos(angle\_in\_radians);

double sin\_value = sin(angle\_in\_radians);

\_x\_coordinate = original\_x\_coordinate \* cos\_value - \_y\_coordinate \* sin\_value;

\_y\_coordinate = original\_x\_coordinate \* sin\_value + \_y\_coordinate \* cos\_value;

}

#pragma endregion

#pragma region Set point color functions

void CPoint::SetPointColor(Color color\_component, double color\_value)

{

switch (color\_component)

{

case RED\_COMPONENT:

\_red\_component = color\_value;

break;

case GREEN\_COMPONENT:

\_green\_component = color\_value;

break;

case BLUE\_COMPONENT:

\_blue\_component = color\_value;

break;

}

}

void CPoint::SetPointRed(double red\_component)

{

\_red\_component = red\_component;

}

void CPoint::SetPointGreen(double green\_component)

{

\_green\_component = green\_component;

}

void CPoint::SetPointBlue(double blue\_component)

{

\_blue\_component = blue\_component;

}

void CPoint::SetPointColor(double red\_component, double green\_component, double blue\_component)

{

SetPointRed(red\_component);

SetPointGreen(green\_component);

SetPointBlue(blue\_component);

}

#pragma endregion

#pragma region Get functions

CPoint CPoint::ClonePoint()

{

CPoint new\_point(GetX(), GetY(), GetZ(), GetRed(), GetGreen(), GetBlue());

return new\_point;

}

double CPoint::GetCoordinate(Axis axis)

{

switch (axis)

{

case X\_AXIS:

return \_x\_coordinate;

break;

case Y\_AXIS:

return \_y\_coordinate;

break;

case Z\_AXIS:

return \_z\_coordinate;

break;

default:

break;

}

}

double CPoint::GetColor(Color color\_component)

{

switch (color\_component)

{

case RED\_COMPONENT:

return GetRed();

break;

case GREEN\_COMPONENT:

return GetGreen();

break;

case BLUE\_COMPONENT:

return GetBlue();

break;

default:

break;

}

}

double CPoint::GetX()

{

return \_x\_coordinate;

}

double CPoint::GetY()

{

return \_y\_coordinate;

}

double CPoint::GetZ()

{

return \_z\_coordinate;

}

double CPoint::GetRed()

{

return \_red\_component;

}

double CPoint::GetGreen()

{

return \_green\_component;

}

double CPoint::GetBlue()

{

return \_blue\_component;

}

void CPoint::DrawPoint()

{

glColor3f(\_red\_component, \_green\_component, \_blue\_component);

glVertex3f(\_x\_coordinate, \_y\_coordinate, \_z\_coordinate);

}

#pragma endregion

**GameState.cpp:**#include "GameState.h"

GameState::GameState()

{

RESET();

GameReboot = true;

}

void GameState::Gameover()

{

xChange = 0;

gunMove = 0;

GameOver = true;

}

void GameState::StageChange()

{

Stage++;

Level\_Number = 1;

xChange\_Multiplier = 1;

}

void GameState::GameFinished()

{

Congratz = true;

xChange = 0;

gunMove = 0;

}

void GameState::Stage2LivePlus1()

{

lives++;

StageUp = true;

}

void GameState::Stage3LivePlus3()

{

lives += 3;

StageUp = true;

}

int GameState::GetLives()

{

return lives;

}

void GameState::SetLives(int lives)

{

this->lives = lives;

}

double GameState::GetXChange()

{

return xChange;

}

void GameState::SetXChange(double x)

{

this->xChange = x;

}

double GameState::GetGunMove()

{

return gunMove;

}

void GameState::SetGunMove(double gm)

{

this->gunMove = gm;

}

bool GameState::GetGameOver()

{

return GameOver;

}

void GameState::SetGameOver(bool GO)

{

this->GameOver = GO;

}

int GameState::GetStage()

{

return Stage;

}

void GameState::SetStage(int St)

{

this->Stage = St;

}

int GameState::GetLevelNum()

{

return Level\_Number;

}

void GameState::SetLevelNum(int LN)

{

this->Level\_Number = LN;

AdvanceLevel();

}

void GameState::AdvanceLevel()

{

if (GetStage() < 3)

SetXCMulti(GetXCMulti() + 4);

else if (GetStage() == 3)

SetXCMulti(GetXCMulti() \* 1.5);

}

double GameState::GetXCMulti()

{

return xChange\_Multiplier;

}

void GameState::SetXCMulti(double xCM)

{

this->xChange\_Multiplier = xCM;

}

bool GameState::GetCongratz()

{

return Congratz;

}

void GameState::SetCongratz(bool C)

{

this->Congratz = C;

}

bool GameState::GetStageUp()

{

return StageUp;

}

void GameState::SetStageUp(bool SU)

{

this->StageUp = SU;

}

bool GameState::GetShotMissed()

{

return ShotMissed;

}

void GameState::SetShotMissed(bool SM)

{

this->ShotMissed = SM;

}

bool GameState::GetBulletHitTarget()

{

return BulletHitTarget;

}

void GameState::SetBulletHitTarget(bool BHT)

{

this->BulletHitTarget = BHT;

}

bool GameState::GetBullsEye()

{

return BullsEye;

}

void GameState::SetBullsEye(bool BE)

{

this->BullsEye = BE;

}

bool GameState::GetGameStarted()

{

return GameStarted;

}

void GameState::SetGameStarted(bool GS)

{

this->GameStarted = GS;

}

bool GameState::GetFirstShotBeenShot()

{

return FirstShotBeenShot;

}

void GameState::SetFirstShotBeenShot(bool FSBS)

{

this->FirstShotBeenShot = FSBS;

}

bool GameState::GetGameReboot()

{

return GameReboot;

}

void GameState::SetGameReboot(bool GGR)

{

this->GameReboot = GGR;

}

bool GameState::GetBBS()

{

return BULLETbeenSHOT;

}

void GameState::SetBBS(bool BULLETBS)

{

this->BULLETbeenSHOT = BULLETBS;

}

void GameState::ShootingHandler()

{

SetGameStarted(true);

if (GetLevelNum() >= 1)

{

SetBBS(true);

}

if (GetLevelNum() == 0)

{

SetLevelNum(GetLevelNum() + 1);

}

// on stage 3 target changes direction with each shot

if (GetStage() == 3)

{

SetXCMulti(GetXCMulti() \* -1);

}

}

void GameState::RESET()

{

if (xChange < 0)

{

xChange \*= -1;

}

GameOver = false;

Congratz = false;

Stage = 1;

lives = 10;

GameStarted = false;

FirstShotBeenShot = false;

Level\_Number = 0;

xChange\_Multiplier = 1;

gunMove = 1;

ShotMissed = false;

BulletHitTarget = false;

BullsEye = false;

StageUp = false;

GameReboot = false;

}

**Gun.cpp:**

#include "Gun.h"

#include <math.h>

#include "CModel.h"

#include "CPolygon.h"

#include "CCircle.h"

Gun::Gun(void)

{

SetNumberOfPolygons(31);

CPolygon square\_in\_cube;

square\_in\_cube.SetNumberOfPoints(4);

square\_in\_cube.SetOnePointCoordinates(0, 0, 0, 1);

square\_in\_cube.SetOnePointCoordinates(1, 0.75, 0, 1);

square\_in\_cube.SetOnePointCoordinates(2, 0.75, 0, 0);

square\_in\_cube.SetOnePointCoordinates(3, 0, 0, 0);

square\_in\_cube.SetPolygonColor(1, 0, 0);

SetOnePolygon(0, square\_in\_cube);

CPolygon square\_in\_cubeA;

square\_in\_cubeA.SetNumberOfPoints(4);

square\_in\_cubeA.SetOnePointCoordinates(0, 0, 1, 0);

square\_in\_cubeA.SetOnePointCoordinates(1, 0, 0, 0);

square\_in\_cubeA.SetOnePointCoordinates(2, 0, 0, 1);

square\_in\_cubeA.SetOnePointCoordinates(3, 0, 1, 1);

square\_in\_cubeA.SetPolygonColor(1, 0, 0);

SetOnePolygon(1, square\_in\_cubeA);

CPolygon square\_in\_cubeB;

square\_in\_cubeB.SetNumberOfPoints(4);

square\_in\_cubeB.SetOnePointCoordinates(3, 0, 0, 1);

square\_in\_cubeB.SetOnePointCoordinates(2, 0, 1, 1);

square\_in\_cubeB.SetOnePointCoordinates(1, 0.75, 1, 1);

square\_in\_cubeB.SetOnePointCoordinates(0, 0.75, 0, 1);

square\_in\_cubeB.SetPolygonColor(1, 0, 0);

SetOnePolygon(2, square\_in\_cubeB);

CPolygon square\_in\_cubeC;

square\_in\_cubeC.SetNumberOfPoints(4);

square\_in\_cubeC.SetOnePointCoordinates(0, 0, 0, 0);

square\_in\_cubeC.SetOnePointCoordinates(1, 0, 1, 0);

square\_in\_cubeC.SetOnePointCoordinates(2, 0.75, 1, 0);

square\_in\_cubeC.SetOnePointCoordinates(3, 0.75, 0, 0);

square\_in\_cubeC.SetPolygonColor(1, 0, 0);

SetOnePolygon(3, square\_in\_cubeC);

CPolygon square\_in\_cubeD;

square\_in\_cubeD.SetNumberOfPoints(4);

square\_in\_cubeD.SetOnePointCoordinates(3, 0.75, 1, 0);

square\_in\_cubeD.SetOnePointCoordinates(2, 0.75, 0, 0);

square\_in\_cubeD.SetOnePointCoordinates(1, 0.75, 0, 1);

square\_in\_cubeD.SetOnePointCoordinates(0, 0.75, 1, 1);

square\_in\_cubeD.SetPolygonColor(1, 0, 0);

SetOnePolygon(4, square\_in\_cubeD);

CPolygon Trigger;

Trigger.SetNumberOfPoints(4);

Trigger.SetOnePointCoordinates(3, 0.2, 1.75, -0.375);

Trigger.SetOnePointCoordinates(2, 0.55, 1.75, -0.375);

Trigger.SetOnePointCoordinates(1, 0.55, 1.35, -0.375);

Trigger.SetOnePointCoordinates(0, 0.2, 1.35, -0.375);

Trigger.SetPolygonColor(1, 1, 1);

SetOnePolygon(5, Trigger);

CPolygon square\_in\_cube1;

square\_in\_cube1.SetNumberOfPoints(4);

square\_in\_cube1.SetOnePointCoordinates(3, 0, 1, 0);

square\_in\_cube1.SetOnePointCoordinates(2, 0.75, 1, 0);

square\_in\_cube1.SetOnePointCoordinates(1, 0.75, 2, 0);

square\_in\_cube1.SetOnePointCoordinates(0, 0, 2, 0);

square\_in\_cube1.SetPolygonColor(1, 0, 0);

SetOnePolygon(6, square\_in\_cube1);

CPolygon square\_in\_cube2;

square\_in\_cube2.SetNumberOfPoints(4);

square\_in\_cube2.SetOnePointCoordinates(0, 0, 1, 1);

square\_in\_cube2.SetOnePointCoordinates(1, 0.75, 1, 1);

square\_in\_cube2.SetOnePointCoordinates(2, 0.75, 1.75, 1);

square\_in\_cube2.SetOnePointCoordinates(3, 0, 1.75, 1);

square\_in\_cube2.SetPolygonColor(0.4, 0.4, 0.4);

SetOnePolygon(7, square\_in\_cube2);

CPolygon square\_in\_cube3;

square\_in\_cube3.SetNumberOfPoints(4);

square\_in\_cube3.SetOnePointCoordinates(0, 0, 1.33, 0);

square\_in\_cube3.SetOnePointCoordinates(1, 0.75, 1.33, 0);

square\_in\_cube3.SetOnePointCoordinates(2, 0.75, 1.33, -0.75);

square\_in\_cube3.SetOnePointCoordinates(3, 0, 1.33, -0.75);

square\_in\_cube3.SetPolygonColor(1, 0, 0);

SetOnePolygon(8, square\_in\_cube3);

CPolygon square\_in\_cube4;

square\_in\_cube4.SetNumberOfPoints(4);

square\_in\_cube4.SetOnePointCoordinates(3, 0, 1, 0);

square\_in\_cube4.SetOnePointCoordinates(2, 0, 1.75, 0);

square\_in\_cube4.SetOnePointCoordinates(1, 0, 1.75, 1);

square\_in\_cube4.SetOnePointCoordinates(0, 0, 1, 1);

square\_in\_cube4.SetPolygonColor(0.4,0.4,0.4);

SetOnePolygon(9, square\_in\_cube4);

CPolygon square\_in\_cube6;

square\_in\_cube6.SetNumberOfPoints(4);

square\_in\_cube6.SetOnePointCoordinates(0, 0.75, 1, 0);

square\_in\_cube6.SetOnePointCoordinates(1, 0.75, 1.75, 0);

square\_in\_cube6.SetOnePointCoordinates(2, 0.75, 1.75, 1);

square\_in\_cube6.SetOnePointCoordinates(3, 0.75, 1, 1);

square\_in\_cube6.SetPolygonColor(0.4,0.4,0.4);

SetOnePolygon(10, square\_in\_cube6);

CPolygon square\_in\_cube7;

square\_in\_cube7.SetNumberOfPoints(4);

square\_in\_cube7.SetOnePointCoordinates(3, 0, 1.33, -0.75);

square\_in\_cube7.SetOnePointCoordinates(2, 0.75, 1.33, -0.75);

square\_in\_cube7.SetOnePointCoordinates(1, 0.75, 1.75, -0.75);

square\_in\_cube7.SetOnePointCoordinates(0, 0, 1.75, -0.75);

square\_in\_cube7.SetPolygonColor(1, 0, 0);

SetOnePolygon(11, square\_in\_cube7);

CPolygon square\_in\_cube8;

square\_in\_cube8.SetNumberOfPoints(4);

square\_in\_cube8.SetOnePointCoordinates(0, 0, 1.75, 0);

square\_in\_cube8.SetOnePointCoordinates(1, 0.75, 1.75, 0);

square\_in\_cube8.SetOnePointCoordinates(2, 0.75, 1.75, -1.5);

square\_in\_cube8.SetOnePointCoordinates(3, 0, 1.75, -1.5);

square\_in\_cube8.SetPolygonColor(0.4,0.4,0.4);

SetOnePolygon(12, square\_in\_cube8);

CPolygon Trigger\_Other\_Side;

Trigger\_Other\_Side.SetNumberOfPoints(4);

Trigger\_Other\_Side.SetOnePointCoordinates(0, 0.2, 1.75, -0.375);

Trigger\_Other\_Side.SetOnePointCoordinates(1, 0.55, 1.75, -0.375);

Trigger\_Other\_Side.SetOnePointCoordinates(2, 0.55, 1.35, -0.375);

Trigger\_Other\_Side.SetOnePointCoordinates(3, 0.2, 1.35, -0.375);

Trigger\_Other\_Side.SetPolygonColor(1, 1, 1);

SetOnePolygon(13, Trigger\_Other\_Side);

CPolygon square\_in\_cube9;

square\_in\_cube9.SetNumberOfPoints(4);

square\_in\_cube9.SetOnePointCoordinates(0, 0, 2, 1);

square\_in\_cube9.SetOnePointCoordinates(1, 0.75, 2, 1);

square\_in\_cube9.SetOnePointCoordinates(2, 0.75, 2, -1.5);

square\_in\_cube9.SetOnePointCoordinates(3, 0, 2, -1.5);

square\_in\_cube9.SetPolygonColor(0.4, 0.4, 0.4);

SetOnePolygon(14, square\_in\_cube9);

CPolygon square\_in\_cube\_7\_OtherSide;

square\_in\_cube\_7\_OtherSide.SetNumberOfPoints(4);

square\_in\_cube\_7\_OtherSide.SetOnePointCoordinates(0, 0, 1.33, -0.75);

square\_in\_cube\_7\_OtherSide.SetOnePointCoordinates(1, 0.75, 1.33, -0.75);

square\_in\_cube\_7\_OtherSide.SetOnePointCoordinates(2, 0.75, 1.75, -0.75);

square\_in\_cube\_7\_OtherSide.SetOnePointCoordinates(3, 0, 1.75, -0.75);

square\_in\_cube\_7\_OtherSide.SetPolygonColor(1, 0, 0);

SetOnePolygon(15, square\_in\_cube\_7\_OtherSide);

CPolygon square\_in\_cube10;

square\_in\_cube10.SetNumberOfPoints(4);

square\_in\_cube10.SetOnePointCoordinates(3, 0, 1.75, -1.5);

square\_in\_cube10.SetOnePointCoordinates(2, 0, 2, -1.5);

square\_in\_cube10.SetOnePointCoordinates(1, 0, 2, 1);

square\_in\_cube10.SetOnePointCoordinates(0, 0, 1.75, 1);

square\_in\_cube10.SetPolygonColor(0.4, 0.4, 0.4);

SetOnePolygon(16, square\_in\_cube10);

CPolygon square\_in\_cube11;

square\_in\_cube11.SetNumberOfPoints(4);

square\_in\_cube11.SetOnePointCoordinates(0, 0.75, 1.75, -1.5);

square\_in\_cube11.SetOnePointCoordinates(1, 0.75, 2, -1.5);

square\_in\_cube11.SetOnePointCoordinates(2, 0.75, 2, 1);

square\_in\_cube11.SetOnePointCoordinates(3, 0.75, 1.75, 1);

square\_in\_cube11.SetPolygonColor(0.4, 0.4, 0.4);

SetOnePolygon(17, square\_in\_cube11);

CPolygon square\_in\_cube12;

square\_in\_cube12.SetNumberOfPoints(4);

square\_in\_cube12.SetOnePointCoordinates(3, 0, 1.75, 1);

square\_in\_cube12.SetOnePointCoordinates(2, 0, 2, 1);

square\_in\_cube12.SetOnePointCoordinates(1, 0.75, 2, 1);

square\_in\_cube12.SetOnePointCoordinates(0, 0.75, 1.75, 1);

square\_in\_cube12.SetPolygonColor(0.4, 0.4, 0.4);

SetOnePolygon(18, square\_in\_cube12);

CPolygon square\_in\_cube13;

square\_in\_cube13.SetNumberOfPoints(4);

square\_in\_cube13.SetOnePointCoordinates(0, 0, 1.75, -1.5);

square\_in\_cube13.SetOnePointCoordinates(1, 0, 2, -1.5);

square\_in\_cube13.SetOnePointCoordinates(2, 0.75, 2, -1.5);

square\_in\_cube13.SetOnePointCoordinates(3, 0.75, 1.75, -1.5);

square\_in\_cube13.SetPolygonColor(0.4, 0.4, 0.4);

SetOnePolygon(19, square\_in\_cube13);

//Sight

CPolygon square\_in\_cube14;

square\_in\_cube14.SetNumberOfPoints(4);

square\_in\_cube14.SetOnePointCoordinates(0, 0.3, 2.1, -1.5);

square\_in\_cube14.SetOnePointCoordinates(1, 0.3, 2, -1.5);

square\_in\_cube14.SetOnePointCoordinates(2, 0.45, 2, -1.5);

square\_in\_cube14.SetOnePointCoordinates(3, 0.45, 2.1, -1.5);

square\_in\_cube14.SetPolygonColor(0, 1, 0);

SetOnePolygon(20, square\_in\_cube14);

CPolygon square\_in\_cube14\_OtherSide;

square\_in\_cube14\_OtherSide.SetNumberOfPoints(4);

square\_in\_cube14\_OtherSide.SetOnePointCoordinates(3, 0.3, 2.1, -1.5);

square\_in\_cube14\_OtherSide.SetOnePointCoordinates(2, 0.3, 2, -1.5);

square\_in\_cube14\_OtherSide.SetOnePointCoordinates(1, 0.45, 2, -1.5);

square\_in\_cube14\_OtherSide.SetOnePointCoordinates(0, 0.45, 2.1, -1.5);

square\_in\_cube14\_OtherSide.SetPolygonColor(0, 1, 0);

SetOnePolygon(21, square\_in\_cube14\_OtherSide);

CCircle bullet\_hole;

bullet\_hole.SetNumberOfPoints(4);

bullet\_hole.MovePolygonCenter(3.7, 18.65, -15.1);

bullet\_hole.SetPolygonColor(0, 0, 0);

bullet\_hole.ScalePolygon(0.1,0.1,0.1);

bullet\_hole.RotateSelfPolygon(Y\_AXIS, 180);

SetOnePolygon(22, bullet\_hole);

CCircle bullet\_hole\_OtherSide;

bullet\_hole\_OtherSide.SetNumberOfPoints(4);

bullet\_hole\_OtherSide.MovePolygonCenter(3.7, 18.65, -15.1);

bullet\_hole\_OtherSide.SetPolygonColor(0, 0, 0);

bullet\_hole\_OtherSide.ScalePolygon(0.1, 0.1, 0.1);

SetOnePolygon(23, bullet\_hole\_OtherSide);

//Nitzra

CPolygon square\_in\_cube15;

square\_in\_cube15.SetNumberOfPoints(4);

square\_in\_cube15.SetOnePointCoordinates(0, 0.3, 2.1, 1);

square\_in\_cube15.SetOnePointCoordinates(1, 0.3, 2, 1);

square\_in\_cube15.SetOnePointCoordinates(2, 0.45, 2, 1);

square\_in\_cube15.SetOnePointCoordinates(3, 0.45, 2.1, 1);

square\_in\_cube15.SetPolygonColor(0.2,0.3,0.4);

SetOnePolygon(24, square\_in\_cube15);

CPolygon square\_in\_cube16;

square\_in\_cube16.SetNumberOfPoints(4);

square\_in\_cube16.SetOnePointCoordinates(0, 0.3, 2.1, 1);

square\_in\_cube16.SetOnePointCoordinates(1, 0.45, 2.1, 1);

square\_in\_cube16.SetOnePointCoordinates(2, 0.45, 2, 0.95);

square\_in\_cube16.SetOnePointCoordinates(3, 0.3, 2, 0.95);

square\_in\_cube16.SetPolygonColor(0.2,0.3,0.4);

SetOnePolygon(25, square\_in\_cube16);

CPolygon Triangle\_Left;

Triangle\_Left.SetNumberOfPoints(4);

Triangle\_Left.SetOnePointCoordinates(3, 0.3, 2.1, 1);

Triangle\_Left.SetOnePointCoordinates(2, 0.3, 2, 1);

Triangle\_Left.SetOnePointCoordinates(1, 0.3, 2, 0.95);

Triangle\_Left.SetOnePointCoordinates(0, 0.3, 2, 0.95);

Triangle\_Left.SetPolygonColor(0.2,0.3,0.4);

SetOnePolygon(26, Triangle\_Left);

CPolygon Triangle\_Right;

Triangle\_Right.SetNumberOfPoints(4);

Triangle\_Right.SetOnePointCoordinates(0, 0.45, 2.1, 1);

Triangle\_Right.SetOnePointCoordinates(1, 0.45, 2, 1);

Triangle\_Right.SetOnePointCoordinates(2, 0.45, 2, 0.95);

Triangle\_Right.SetOnePointCoordinates(3, 0.45, 2, 0.95);

Triangle\_Right.SetPolygonColor(0.2,0.3,0.4);

SetOnePolygon(27, Triangle\_Right);

CPolygon square\_in\_cube17;

square\_in\_cube17.SetNumberOfPoints(4);

square\_in\_cube17.SetOnePointCoordinates(3, 0, 1.75, 1);

square\_in\_cube17.SetOnePointCoordinates(2, 0.75, 1.75, 1);

square\_in\_cube17.SetOnePointCoordinates(1, 0.75, 1.75, -1.5);

square\_in\_cube17.SetOnePointCoordinates(0, 0, 1.75, -1.5);

square\_in\_cube17.SetPolygonColor(0.4, 0.4, 0.4);

SetOnePolygon(28, square\_in\_cube17);

CPolygon Bottom\_Of\_Gun;

Bottom\_Of\_Gun.SetNumberOfPoints(4);

Bottom\_Of\_Gun.SetOnePointCoordinates(0, 0, 0, 0);

Bottom\_Of\_Gun.SetOnePointCoordinates(1, 0.75, 0, 0);

Bottom\_Of\_Gun.SetOnePointCoordinates(2, 0.75, 0, 1);

Bottom\_Of\_Gun.SetOnePointCoordinates(3, 0, 0, 1);

Bottom\_Of\_Gun.SetPolygonColor(1,0,0);

SetOnePolygon(29, Bottom\_Of\_Gun);

CPolygon Bottom\_Of\_Trigger;

Bottom\_Of\_Trigger.SetNumberOfPoints(4);

Bottom\_Of\_Trigger.SetOnePointCoordinates(3, 0, 1.33, 0);

Bottom\_Of\_Trigger.SetOnePointCoordinates(2, 0.75, 1.33, 0);

Bottom\_Of\_Trigger.SetOnePointCoordinates(1, 0.75, 1.33, -0.75);

Bottom\_Of\_Trigger.SetOnePointCoordinates(0, 0, 1.33, -0.75);

Bottom\_Of\_Trigger.SetPolygonColor(1, 0, 0);

SetOnePolygon(30, Bottom\_Of\_Trigger);

}

**main.cpp:**#include "CMyOpenGLInit.h"

void main(int argc, char\* argv[])

{

CMyOpenGLInit MyGame(argc, argv);

}

**Target.cpp:**#include "Target.h"

#include <stdio.h>

Target::Target()

{

SetNumberOfPolygons(5);

CCircle circle5(360);

circle5.ScaleSelfPolygon(1.3, 1.3, 1.3);

circle5.SetPolygonColor(1, 0, 0);

circle5.MovePolygonCenter(-0.142, 0.7, -15);

SetOnePolygon(0, circle5);

circle5.ScaleSelfPolygon(0.8, 0.8, 0.8);

circle5.SetPolygonColor(1, 1, 1);

circle5.MovePolygon(0, 0, 0.0001);

SetOnePolygon(1, circle5);

circle5.ScaleSelfPolygon(0.7, 0.7, 0.7);

circle5.SetPolygonColor(1, 0, 0);

circle5.MovePolygon(0, 0, 0.0001);

SetOnePolygon(2, circle5);

circle5.ScaleSelfPolygon(0.6, 0.6, 0.6);

circle5.SetPolygonColor(1, 1, 1);

circle5.MovePolygon(0, 0, 0.0001);

SetOnePolygon(3, circle5);

circle5.ScaleSelfPolygon(0.5, 0.5, 0.5);

circle5.SetPolygonColor(0.4, 0.4, 0.4);

circle5.MovePolygon(0, 0, 0.0001);

SetOnePolygon(4, circle5);

}

void Target::TargetMovement(GameState &gs1)

{

if (gs1.GetGameReboot() == true)

{

gs1.SetXChange(0.0025);

gs1.SetGameReboot(false);

}

if (gs1.GetStage() == 1 && gs1.GetBBS() == false)

{

if (GetPolygon(0).GetPolygonCenter().GetCoordinate(X\_AXIS) > 10.4)

{

gs1.SetXChange(gs1.GetXChange() \* -1);

}

if (GetPolygon(0).GetPolygonCenter().GetCoordinate(X\_AXIS) < -10.4)

{

gs1.SetXChange(gs1.GetXChange() \* -1);

}

MoveModel(gs1.GetXChange() \* gs1.GetXCMulti(), 0, 0);

}

//Target Movement While The Bullet is Being shot:

if (gs1.GetStage() > 1)

{

if (GetPolygon(0).GetPolygonCenter().GetCoordinate(X\_AXIS) > 10.4)

{

gs1.SetXChange(gs1.GetXChange() \* -1);

}

if (GetPolygon(0).GetPolygonCenter().GetCoordinate(X\_AXIS) < -10.4)

{

gs1.SetXChange(gs1.GetXChange() \* -1);

}

MoveModel(gs1.GetXChange() \* gs1.GetXCMulti(), 0, 0);

}

}

Target::~Target()

{

}