# 

# **עבודת גמר**

# **לקבלת תואר טכנאי**

# **הנדסת תוכנה**

**נושא הפרויקט:** SLENDER

**שפת תכנות:** Java

**סביבת העבודה** ‎Eclipse

**שם המגיש:** אופק קזס

**ת"ז:** 314960568

**מכללה:** אורטחולון

**כיתה:** י"ג 2, מגמת הנדסת תוכנה

**מנחה:** מיקי שרוני

**מאי 2015 תשע"ה**

**תוכן עניינים**

**הצעת פרויקט ............................................................................................. 4 - 3**

1. **מבוא ......................................................................................................... 5**
   1. **מטרה .................................................................................................................................** 5
   2. **תיאור המערכת .................................................................................................................... 5**
   3. **מבנה המסמך ........................................................................................................................ 5**
   4. **ביבליוגרפיה ........................................................................................................................** 5
2. **מפרטי תוכנה ......................................................................................... 6**-**8**
   1. **תיאור כללי .......................................................................................................................** 6
   2. **ממשק אדם מכונה MMI .................................................................................................** 7
   3. **תיאור אלגוריתמים .............................................................................................................** 8

1. **תכנון .................................................................................................. 9-19**
   1. **חלוקה למודולים .................................................................................................................** 9
   2. **פירוט המודולים+רשימת נתונים ......................................................................................** 10-19
   3. **מבנה נתונים .......................................................................................................................** 19
2. **מדריך למשתמש ....................................................................................... 20**
   1. **הוראות התקנה ( Setup ) ..............................................................................................** 20
   2. **הוראות הפעלה ...................................................................................................................** 20
   3. **הודעות מערכת ...................................................................................................................** 20

**5. נספח - קוד מקור .................................................................................... 21-76**

**הצעת נושא לפרויקטים לטכנאים במגמת הנדסת תוכנה**

א. **הנושא:** SLENDER

ב. **שם המנחה:** חן אולמר

ג. **שם התלמיד:**  **אופק קזס ת.ז:** 314960568

ד. **עבודה ביחיד**

ה. **אופי עבודת הפרויקט:** תכנון ובניית אב טיפוס.

ו. **מקום ביצוע הפרויקט:** מכללת אורט חולון

ז. **תיאור נושא המשחק:**

המשחק Slender מתנהל ביער בחצות, אדם כלשהו שהוא השחקן נרדף על ידי ישות אנונימית ששמה מוכר רק כסלנדר, ביער פזורים 8 דפים שהשחקן צריך למצוא על מנת לנצח את הסלנדר וכדי שהוא ישחרר אותך.

כאשר השחקן אוסף דף, רמת הקושי עולה ואיתה המתח, הסלנדר הופך ונהיה עצבני על כל דף שנאסף ולכן רודף אחריך יותר מהר.

את המשחק אסביר בכמה שלבים:

1. הכרת המקום: השחקן מוצא עצמו לראשונה ביער חשוך, זו ההזדמנות ללמוד את המקום (שנבנה רנדומלית כל פעם). לפני שהשחקן אוסף את הדף הראשון הסלנדר לא רודף אחריו, למרות שאם עבר יותר מידי זמן ואף דף עוד לא נאסף, הסלנדר יתחיל לרדוף

2. מהלך המשחק: השחקן צריך למצוא כמה שיותר מהר 8 דפים שפזורים על העצים במשחק, לשחקן יש פנס, ולפנס סוללה שנגמרת כל שניה שעוברת, אם סוללת הפנס נגמרה השחקן יצטרך למצוא עצמו בחשכה.

על כל דף שנמצא רמת הקושי עולה והסלנדר מתעצבן, כעת הוא ישתגר למקום הקרוב לשחקן באופן מהיר יותר.

אם השחקן התקרב לסלנדר יותר מידי או שהסלנדר מצא אותו, השחקן מת והסלנדר ניצח, לעומת זאת, אם השחקן מצא את שמונת הדפים לפני השסלנדר תפס אותו, ברכותיי, הוא ניצח במשחק

ח. **פירוט הדרישות מהמערכת:**

* המערכת תנהל את המשחק של השחקן ושל היריב הממוחשב.
* המערכת תטען את המפה בתחילת שלב.
* המערכת תחשב את מיקום השחקן ולפיו תעדכן את מיקום הסלנדר.
* המערכת תבצע בדיקות תקינות כגון: האם השחקן אסף את כל הדפים. האם השחקן פוגע בקירות המבנים שנמצאים ביער (והמערכת תגיב בהתאם), האם הסוללות של הפנס של השחקן התרוקנו.
* המערכת תכריז על הודעות ניצחון ועל הניקוד בעקבותיו במידה והשחקן מצא את כל הדפים. המערכת תודיע על הפסד במידה והסלנדר תפס את השחקן לפני שמצא את כל הדפים.
* המערכת לא תאפשר לחדור או לעבור בין קירות המפה, העצים, המבנים או המכשולים בה. מלבד זאת התנועה במשחק היא חופשית.
* מבנה הנתונים עליו ייבדקו תנאי המשחק יהיה גרפים.
* אלגוריתם הסלנדר יממומש על ידי אלגוריתם שישגר את הסלנדר, בזמנים שונים שנקבעים על ידי דרגת הקושי, אל מקום הקרוב אל השחקן, אין בעיה שהסלנדר ייתקע במבנה כלשהו מכיוון שבדיקת הפגיעות עובדת כך שאם אחד מהאובייקטים שנבדקים פוגע באחר, האובייקט זז למקום שלא תהיה פגיעה. לא השתמשתי באלגוריתם ידוע כמו דייקסטרה מכיוון שזהו אלגוריתם למציאת המסלול הקצר ביותר, אך לפי המשחק המקורי של הסלנדר שאני מתבסס עליו, הסלנדר משתגר ממקום למקום ולא זז בצעדים קטנים, כך שהייתי יכול לקבוע את מיקום הסלנדר רק לפי מיקום השחקן.

ט. **מפרט טכני:**

מעבד: מעבד Pentium 4 2000 MHz.

זיכרון: RAM512.

כרטיס מסך: תמיכה בOpenGL 1.3

סביבת עבודה: WINDOWS 7.

תוכנה: Java.

י. **לוח זמנים לביצוע הפרויקט לפי שלבים:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 11/2014 | 12/2014 | 01/2015 | 02/2015 | 03/2015 | 04/2015 | 05/2015 |
| הכנת הצעת הפרויקט | X |  |  |  |  |  |  |
| הגשת ההצעה | X |  |  |  |  |  |  |
| תכנון |  | X |  |  |  |  |  |
| גרפיקה |  | X |  |  |  |  |  |
| כתיבה |  |  | X | X | X |  |  |
| בדיקה |  |  |  |  | X | X |  |
| הפעלה והגשה |  |  |  |  |  | X | X |

# **1. מבוא:**

**1.1 מטרה:**

מטרת המשחק היא למצוא את כל שמונת הדפים לפני שהסלנדר מוצא אותך

**1.2 תיאור המערכת:**

המערכת מורכבת מיער שעליו יש עצים שעליהם דפים, שחקן שעליו פנס וסלנדר

**1.3 מבנה המסמך:**

הספר כולל ארבעה פרקים:

הפרק הראשון- מבוא למשחק עצמו: מטרת המשחק, תיאור כללי של המשחק, מבנה המסמך כולו ורשימת מקורות עזר בכתיבת הפרויקט.

הפרק השני- פירוט התוכנה עצמה. תיאור מפורט של המשחק, ממשק אדם-מכונה, יכולות המערכת, תיאור האלגוריתם ורשימת נתונים.

הפרק השלישי- פירוט תכנון המערכת. תיאור המודולים השונים ומבנה הנתונים.

הפרק הרביעי- הוראות הפעלה. פירוט הוראות התקנה והפעלה.

והפרק האחרון – מכיל את כל קוד המקור של הפרויקט

**1.4 רשימה ביבליוגרפית:**

\* http://docs.oracle.com/javase/tutorial/ מדריך לג'אווה

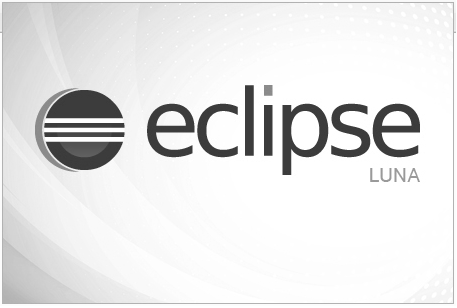
\* http://stackoverflow.com/ פורום תמיכה בג'אווה

\*חומר שנלמד בשיעורים בתחילת השנה

\*http://www.opengl-tutorial.org/אתר ללמידת ספריית openGl

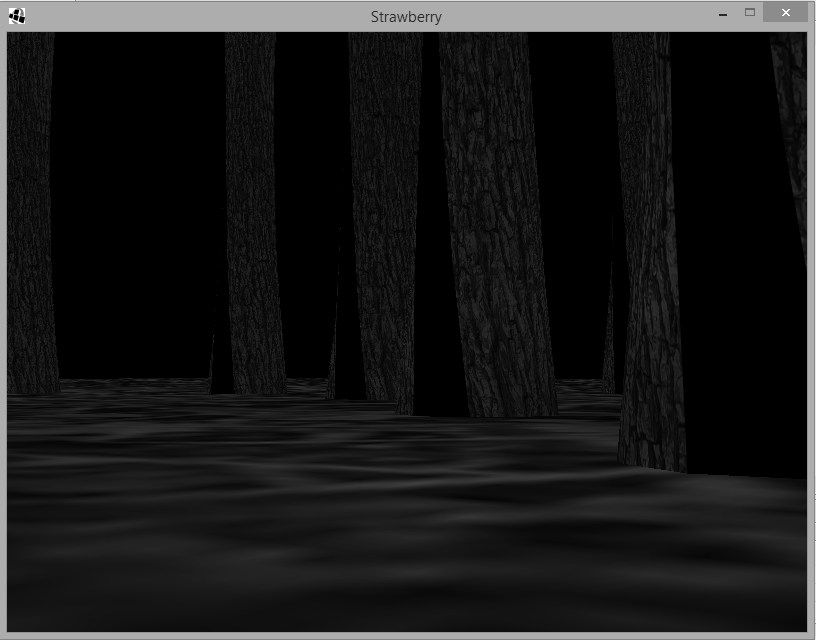
# **2. מפרטי תוכנה:**

ישום זה פותח בסביבת העבודה :



**2.1 תיאור כללי:**

המשחק מתחיל כאשר השחקן נמצא ביער, מחזיק פנס,, יש עצים ועליהם 8 דפים



**2.1.2 מצב של סיום משחק:**

המשחק יכול להסתיים מכמה סיבות :

* + - 1. השחקן אסף את כל הדפים
      2. הסלנדר תפס את השחקן

**2.1.3 אסטרטגית המשחק:**

האסטרטגייה היא דבר ראשון, לא להביט לאחור ולא לעשות בשום אופן פניות חדות, לרוץ רק כאשר יש ידיעה שהסלנדר קרוב, הפנס גם מקרב את הסלנדר יותר, ולכן עדיף לא להשתמש בפנס הרבה, ללכת ברוב המקרים ישר ואם הגעת לסוף המפה בעדינות ובאיטיות יחסית לסובב את כיוון ההליכה (בזהירות, הוא עוקב), ואחרי איסוף דף, לרוץ.

**2.2 ממשק אדם מכונה MMI :1**

1

2

3

4

1. כפתור התחלת משחק חדש
2. כפתור להצגת חלון תיאור המשחק וחוקיו.
3. כפתור אפשרויות
4. כפתור יציאה מן המשחק
   1. **תיאור אלגוריתמים :**

במשחק זה קיים אלגוריתם אשר לפי דרגת הקושי (שנקבעת לפי מספר הדפים שנאספו), משגר בזמנים שונים את האויב הראשי של השחקן, הסלנדר, למקום שהוא מאחוריי השחקן, כך שככל שהשחקן אוסף יותר דפים, כך הסלנדר משתגר במהירות יותר גדולה למיקומו הקרוב של השחקן וגורם לשחקן למהר על מנת לא להיתפס

האלגוריתם:

1. נרוץ על עץ האובייקטים
   1. נבדוק קלט

1.1.1) במידה והשחקן לחץ על מקש כלשהו הוסף אותו לרשימת ההקשות ושמור את מיקום העכבר

1.1.2) במידה והשחקן הקיש במקלדת או בעכבר הזז את השחקן

* + 1. אחרת הקטן את התאוצה
  1. נבדוק פגיעה בין השחקן לעץ הנוכחי
     1. במידה ויש פגיעה, הפוך את כיוון ההליכה של השחקן, מה שיגרום לו לעצור
        1. ) במידה ויש על העץ דף והשחקן הקיש בעכבר, אסוף אותו ועלה את רמת הקושי
  2. במידה והשחקן קרוב מידי לסלנדר הראה הודעת אזהרה, ובמידה שהוא קרוב מידי סיים את המשחק
  3. במידה והשחקן הקיש F הפעל או בטל את הפנס

1.4.1) במידה והפנס מופעל הורד מהעוצמה שלו

1.4.1.1) במידה והעוצמה של הפנס חלשה מידי השהה אותה

1. לאחר ביצוע הלולאה הסלנדר יעדכן את הטיימר, במידה והטיימר יגיע לערך המבוקש הסלנדר ישתגר ביחס למיקום השחקן
2. במידה והשחקן קרוב מידי לסלנדר לתקופת זמן גדולה מידי, סיים את המשחק
3. במידה והשחקן התנגש בגבולות המפה, עצור אותו
4. נרוץ על רשימת האובייקטים פעם נוספת

4.1) הוסף את מנוע הרינדור לאובייקט

4.2) צייר את האובייקט על המסך בנוסף תאורה כללית

4.3) צייר את האובייקט פעם נוספת בתוספת התאורות שנמצאות בסצנה (כמו הפנס)

6) במידה והשחקן מצא 8 דפים, סיים את המשחק

7) במידה ולא חזור על האלגוריתם

סיבוכיות זמן O(n)

**3. תכנון:**

**3.1 חלוקה למודלים :**

1. **מבנה אובייקט משחק:**

מבנה המשמש לניהול, תכנון והצגה של מודל

1. **מבנה מאפיין משחק:**

מייצג פונקציונליות מסוימת לאובייקט המשחק

1. **מבנה מצלמה:**

מציג את המשחק על גבי המסך ביחס למיקום, והסיבוב, כאשר המצלמה היא מאפיין משחק

1. **מבנה עצים:**

מחזיק ברשימות העצים שיוצגו במשחק, כאשר כל עץ הוא אובייקט משחק

1. **מבנה רצפה:**

מחזיק ומציג את האדמה שעליה השחקן יזוז, הסלנדר והעצים יונחו עליה, המבנה יורש מאובייקט משחק

1. **מבנה פנס:**

מכיל אובייקט משחק ומאפיין משחק, יורש ממחלקת אור נקודתי ונחשב כבן למצלמה בגרף הראשי

1. **מבנה גרף ראשי/עץ אשי:**

מכיל בתוכו את שורש העץ הראשי שממנו ניתן להגיע אל כל האובייקטים והמאפיינים שלהם במשחק

1. **מבנה דפים:**

מכיל רשימה של דפים, שנמצאים כבנים לרשימת העצים ומיקומם נמצא ביחס לעצים, כאשר כל דף הוא אובייקט משחק

1. **מבנה סלנדר:**

מבנה שמשמש לניהול האויב הראשי, ובו אובייקט (שבו מאפיין להצגת המודל) וסוג החומר שעליו תושפע התאורה.

1. **מבנה התנהלות המשחק:**

מבנה הבודק מצב השחקן במשחק, האם הוא לא חורג מגבולות המפה, בודק האם יש מצב ניצחון והפסד ועוד

**3.2 פירוט המודלים ורשימת הנתונים:**

מבנה CoreEngine

**Private Boolean** \_isRunning //האם המנוע רץ

**Private Game** \_game//המשחק

**private int** \_width//אורך המסך

**private int** \_height//גובה המסך

**private double** \_frameTime//זמן כל פריים

**private RenderingEngine** \_renderingEngine//מנוע הציור

**שם הפונקציה:**  **public** CoreEngine(**int** width, **int** height, **double** frameRate, Game game)

המטודה בונה את המנוע

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** createWindow()

המטודה בונה חלון ומנוע ציור חדש

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** start()

המטודה מתחילה להריץ את המנוע

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** stop()

המטודה מפסיקה את ריצת המנוע

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** run()

המטודה יוצרת משחק חדש, מעדכנת אותו כל פריים ומציירת אותו

סיבוכיות: o(n)

**שם הפונקציה:**  **public** **static** **void** cleanUp()

המטודה מנקה ומוחקת את כל המידע

סיבוכיות: o(1)

מבנה Game

**Private GameObject** \_root //שורש העץ הראשי

**שם הפונקציה:**  **public** Game()

המטודה בונה את המשחק

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** init()

המטודה מאתחלת את המשחק

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** input(float delta)

המטודה מקבלת קלט מהמשתמש

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** update(float delta)

המטודה מעדכנת כל פריים את המשחק

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** render(RenderingEngine engine)

המטודה מציירת את המשחק

סיבוכיות: o(n)

**שם הפונקציה:**  **public** **void** cleanUp()

המטודה מנקה ומוחקת את כל המידע

סיבוכיות: o(1)

מבנה GameObject

**Private ArrayList<GameObject> \_children //בנים של האובייקט**

**Private ArrayList<GameComponent> \_components //מאפייני האובייקט**

**Private Transform \_transform //ההגדרות הפיזיקליות של האובייקט  
private Boolean \_isImportant//האם צריך לצייר את האובייקט כל פריים**

**שם הפונקציה:**  **public** GameObject()

המטודה בונה את האובייקט

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** addChild(GameObject o)

המטודה מוסיפה אובייקט לרשימת הבנים

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** addComponent(GameComponent c)

המטודה מוסיפה מאפיין לרשימת המאפיינים

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** input(float delta)

המטודה מקבלת קלט

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** update(float delta)

המטודה מעדכנת כל פריים את האובייקט

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** render(Shader shader, RenderingEngine engine, Vector3f position)

המטודה מציירת את האובייקט

סיבוכיות: o(n)

**שם הפונקציה:**  **public** **void** addToRenderingEngine(RenderingEngine engine)

המטודה מוסיפה את האובייקט למנוע הציור

סיבוכיות: o(1)

מבנה Input

**Private static Boolean[] \_lastKeys //המקשים שנלחצו (לפי מספר המקש)**

**Private static Boolean[] \_lastMouse //מקשי העכבר שנלחצו**

**שם הפונקציה:**  **public** **void** update()

המטודה בודקת ומוסיפה לרשימות איזה מקשים נלחצו

סיבוכיות: o(n)

**שם הפונקציה:**  **public** **static** **boolean** getKey(int keyCode)

המטודה בודקת ומחזירה האם המקש נלחץ

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **static** **boolean** getMouse(int mouseButton)

המטודה בודקת ומחזירה האם המקש נלחץ

סיבוכיות: o(1)

מבנה Time

**Private static final long**  SECOND //שניה אחת

**שם הפונקציה:**  **public** **static** **double** getTime()

המטודה מחזירה את הזמן שעבר מתחילת המשחק

סיבוכיות: o(1)

מבנה Transform

**Private Transform \_parent //האבא**

**Private Matrix4f \_parentMatrix //מטריקס האב**

**Private Vector3f \_position //המיקום הנוכחי**

**Private Quaternion \_rotation //הסיבוב הנוכחי**

**Private Vector3f \_scale //הגודל הנוכחי**

**Private Vector3f \_oldPosition //המיקום הקודם**

**Private Quaternion \_oldRotation //הסיבוב הקודם**

**Private Vector3f \_oldScale //הגודל הקודם**

**שם הפונקציה:**  **public** Transform()

המטודה בונה את ההגדרות

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** update()

המטודה מעדכנת כל פריים את ההגדרות

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** rotate(Vector3f axis, float angle)

המטודה מסובבת את המחלקה

סיבוכיות: o(1)

**שם הפונקציה:**  **private** **Boolean** hasChanged()

המטודה בודקת האם המיקום, סיבוב והגודל השתנו

סיבוכיות: o(1)

מבנה GameComponent

**Private GameObject \_parent //האבא**

**שם הפונקציה:**  **public** **void** input(float delta)

המטודה מקבלת קלט

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** update(float delta)

המטודה מעדכנת כל פריים את המאפיין

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** render(Shader shader, RenderingEngine engine)

המטודה מציירת את המאפיין

סיבוכיות: o(1)

**שם הפונקציה:**  **private** **void** addToRenderingEngine(RenderingEngine engine)

המטודה מוסיפה את המאפיין למנוע אם צריך

סיבוכיות: o(1)

מבנה Camera

**Private Boolean \_mouse //מגדיר שליטה באמצעות העכבר**

**Public Matrix4f \_projection //האזור שעליו יצויר המשחק**

**Private Vector3f \_movementVector //הציר שהמצלמה תזוז בו**

**Private Vector3f collisionVector //באיזה ציר מתבצעת פגיעה**

**Private Vector3f \_oldPos //המיקום הישן**

**Private Vector3f \_newPos //המיקום החדש**

**שם הפונקציה:**  **public** Camera(float fov, float aspectRatio, float zNear, float zFar)

המטודה בונה את המצלמה

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** input(float delta)

המטודה מקבלת את הקלט ומציבה בווקטורים המתאימים

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** update(float delta)

המטודה מעדכנת המיקום והסיבוב של המצלמה

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** move(Vector3f direction, float amount)

המטודה מזיזה את המצלמה

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** addToRenderingEngine(RenderingEngine engine)

המטודה מוסיפה את המאפיין למנוע הציור

סיבוכיות: o(1)

מבנה BaseLight

**Private Vector3f \_color //צבע התאורה**

**Private float \_intensity //חוזק התאורה**

**Private Shader \_shader //אפקט התאורה**

**שם הפונקציה:**  **public** BaseLight(Vector3f color, float intensity)

המטודה בונה את התאורה

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** addToRenderingEngine(RenderingEngine engine)

המטודה מוסיפה את התארה למנוע הציור

סיבוכיות: o(1)

מבנה DirectionalLight

**שם הפונקציה:**  **public** DirectionalLight(Vector3f color, float intensity)

המטודה בונה את המצלמה

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **Vector3f** getDirection()

המטודה מחזירה את כיוון התאורה הישירה

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** addToRenderingEngine(RenderingEngine engine)

המטודה מוסיפה את התאורה למנוע הציור

סיבוכיות: o(1)

מבנה PointLight

**Private float \_range //טווח מקסימלי לתאורה**

**Private Vector3f \_attenuation //ההדללה של התאורה**

**שם הפונקציה:**  **public** PointLight(Vector3f color, float intensity, Vector3f attenuation)

המטודה בונה את התאורה

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** addToRenderingEngine(RenderingEngine engine)

המטודה מוסיפה את המאפיין למנוע הציור

סיבוכיות: o(1)

מבנה SpotLight

**Private float \_cutoff**

**שם הפונקציה:**  **public** SpotLight(Vector3f color, float intensity, Vector3f attenuation, float cutoff)

המטודה בונה את התאורה

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** addToRenderingEngine(RenderingEngine engine)

המטודה מוסיפה את המאפיין למנוע הציור

סיבוכיות: o(1)

מבנה MeshRenderer

**Private Mesh \_mesh //המודל**

**Private Material \_material //סוג החומר**

**שם הפונקציה:**  **public** ModelRenderer(Mesh mesh, Material material)

המטודה בונה את התאורה

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** render(Shader shader, RenderingEngine engine)

המטודה מציירת מודל (מאפיין)

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** addToRenderingEngine(RenderingEngine engine)

המטודה מוסיפה את המאפיין למנוע הציור

סיבוכיות: o(1)

מבנה ForwardAmbient

**שם הפונקציה:**  **public** ForwardAmbient()

המטודה בונה את אפקט התאורה הכללית בטכנולוגיות Forward Rendering

סיבוכיות: o(n)

**שם הפונקציה:**  **public** **void** updateUniforms(Transform transform, Matetial material, RenderingEngine engine)

המטודה מעדכנת את התאורה הכללית

סיבוכיות: o(1)

מבנה ForwardDirectional

**שם הפונקציה:**  **public** ForwardDirectional()

המטודה בונה את אפקט התאורה הישירה בטכנולוגיות Forward Rendering

סיבוכיות: o(n)

**שם הפונקציה:**  **public** **void** updateUniforms(Transform transform, Matetial material, RenderingEngine engine)

המטודה מעדכנת את התאורה הישירה

סיבוכיות: o(1)

מבנה ForwardPoint

**שם הפונקציה:**  **public** ForwardPoint()

המטודה בונה את אפקט התאורה בטכנולוגיות Forward Rendering

סיבוכיות: o(n)

**שם הפונקציה:**  **public** **void** updateUniforms(Transform transform, Matetial material, RenderingEngine engine)

המטודה מעדכנת את התאורה הכללית

סיבוכיות: o(1)

מבנה ForwardSpot

**שם הפונקציה:**  **public** ForwardSpot()

המטודה בונה את אפקט התאורה בטכנולוגיות Forward Rendering

סיבוכיות: o(n)

**שם הפונקציה:**  **public** **void** updateUniforms(Transform transform, Matetial material, RenderingEngine engine)

המטודה מעדכנת את התאורה הכללית

סיבוכיות: o(1)

מבנה Material

**Private HashMap<String, Texture> \_textureHash //רשימת הטקסטורות לחומר**

**Private HashMap<String, Vector3f> \_vectorHash //רשימת וקטורים לחומר**

**Private HashMap<String, Float> \_floatHash //רשימת משתנים לחומר**

**שם הפונקציה:**  **public** Material()

המטודה בונה סוג החומר

סיבוכיות: o(1)

מבנה Mesh

**Private int \_size //גודל המודל**

**Private int \_ibo //באפר הנקודות של המודל**

**Private int \_vbo //באפר הקווים המנחים של המודל**

**שם הפונקציה:**  **public** Mesh(String filename)

המטודה בונה את המודל בהתאם לקובץ

סיבוכיות: o(n \* n)

**שם הפונקציה:**  **public** **void** render()

המטודה מציירת את המודל

סיבוכיות: o(1)

מבנה RenderingEngine

**Private Camera \_mainCamera //המצלמה שעליה יוציירו כל האובייקטים**

**Private Vector3f \_ambientLight //התאורה הכללית של הסצנה**

**Private ArrayList<BaseLight> \_lights //רשימת התאורות  
private BaseLight \_activeLight //התאורה שבדיוק מצויירת**

**שם הפונקציה:**  **public** RenderingEngine()

המטודה בונה את מנוע הציור

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** render(GameObject object)

המטודה מציירת את האובייקט על המסך

סיבוכיות: o(n \* n)

**שם הפונקציה:**  **public** **void** clrScreen()

המטודה מנקה את המסך מהפריים הקודם

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** initializeGL2D()

המטודה מכינה את הגרפיקה הדו מימדית

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** initializeGL3D()

המטודה מכינה את הגרפיקה התלת מימדית

סיבוכיות: o(1)

מבנה Shader

**Private int \_program //הקובץ של האפקט**

**Private HashMap<String, Integer> \_uniforms //רשימת המשתנים של האפקט**

**שם הפונקציה:**  **public** Shader()

המטודה בונה את האפקט

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **String** loadShader(String filename)

המטודה טוענת את הקובץ

סיבוכיות: o(n)

מבנה Texture

**Private int \_id //האינדקס של הטקסטורה על כרטיס המסך/ראם**

**שם הפונקציה:**  **public** Texture(int id)

המטודה בונה את הטקסטורה

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** bind()

המטודה מצרפת את הטקסטורה למנהל של openGl

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **static Texture** loadTextureID(String filename)

המטודה טוענת טקסטורה מקובץ

סיבוכיות: o(n)

מבנה Vertex

**Private static final SIZE //גודל הנקודה**

**Private Vector3f \_position //מיקום הנקודה**

**Private vector2f \_textCoords //מיקום לטקסטורה**

**Private Vector3f \_normal //מיקום הנורמל**

**שם הפונקציה:**  **public** Vertex(Vector3f position, Vector2f texture, Vector3f normal)

המטודה בונה את הנקודה

סיבוכיות: o(1)

מבנה Window

**שם הפונקציה:**  **public** static **void** createWindow(int width, int height, String title)

המטודה בונה חלון חדש

סיבוכיות: o(n)

**שם הפונקציה:**  **public** **static void** render()

המטודה גורמת למסך להתעדכן

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** dispose()

המטודה סוגרת את החלון ומוחקת את כל ההגדרות

סיבוכיות: o(1)

מבנה IndexedModel

**Private ArrayList<Vector3f> \_positions //מיקומים**

**Private ArryaList<Vector2f> \_textCoords //מיקומי טקסטורות**

**Private ArrayList<Vector3f> \_normals //רשימת נורמלים  
ArrayList<Integer> \_indices //רישמת נקודות**

**שם הפונקציה:**  **public** IndexedModel()

המטודה בונה את המודל

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** calcNormals()

המטודה מחשבת את הנורמלים של המודל

סיבוכיות: o(n)

מבנה OBJindex

**Public int \_vertexIndex //הנקודה**

**Public int \_textCoordIndex //הטקסטורה**

**Public int \_normalIndex //הנורמל**

**שם הפונקציה:**  **public** **boolean** equals(Object obj)

המטודה בודקת האם האובייקט שהתקבל שווה לנוכחי

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **int** hashCode()

המטודה מחשבת את מספר המודל

סיבוכיות: o(1)

מבנה OBJModel

**Private ArrayList<Vector3f> \_positions //מיקומים**

**Private ArrayList<Vector2f> \_textCoordinates //מיקומי טקסטורות**

**Private ArrayList<Vector3f> \_normals //נורמלים  
private ArrayList<OBJIndex> \_indices //נקודות**

**Private Boolean \_hasTextCoords //האם יש טקסטורה למודל**

**Private Boolean \_hasNormals //האם יש למודל נורמלים**

**שם הפונקציה:**  **public** OBJModel(string filename)

המטודה טוענת המודל

סיבוכיות: o(n)

**שם הפונקציה:**  **public** **IndexdModel** toIndexedModel()

המטודה הופכת את המודל לשימוש בטכנולוגיית ציור יעילה

סיבוכיות: o(n)

**שם הפונקציה:**  **public** **OBJIndex** parseOBJIndex(String token)

המטודה מוסיפה, אם צריך, למודל טקסטורות ונורמלים

סיבוכיות: o(1)

מבנה Arrange

**שם הפונקציה:**  **public** **static ArrayList<GameObject>** arrangeTrees(GameObject parent, int terrainWidth, int terrainDepth)

המטודה מחשבת ומייצרת עצים ומיקום לעצים על המסך באופן נרדומלי

סיבוכיות: o(n \* n)

מבנה FinalGame

**private** GameObject cameraObject; //אובייקט המצלמה

**private** Camera camera; //המצלמה

**private** SpotLight flashlightComponent; //מאפיין הפנס

**private** GameObject flashLight; //אובייקט הפנס

**private** **boolean** isFlashOn; //האם הפנס מופעל

**private** **float** flashIntensity; //עוצמת תאורת הפנס

**private** GameObject terrain; //הרצפה

**private** Material forestGrass; //חומר הרצפה

**private** **float** terrainDepth = 1000.0f; //עומק הרצפה

**private** **float** terrainWidth = 1000.0f; //רוחב הרצפה

**private** ArrayList<GameObject> trees; //רשימת עצים

**private** **int** difficulty; //רמת הקושי

**שם הפונקציה:**  **public** FinalGame()

המטודה בונה את מנוע הציור

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** init()

המטודה מאתחלת את כל האובייקטים והמפאיינים של המשחק

סיבוכיות: o(n \* n)

**שם הפונקציה:**  **public** **void** input(float delta)

המטודה מקבלת פלט

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** update(float delta)

המטודה מעדכנת את כל האובייקטים והמאפיינים של המשחק

סיבוכיות: o(1)

מבנה Page

**Private static ArrayList<GameObject> \_pages //רשימת דפים**

**Private static Material \_material //חומר הדפים**

**Private static Mesh \_model //מודל הדפים**

**שם הפונקציה:**  **public** **void** init(ArrayList<GameObject> trees)

המטודה בונה את הדפים ומוסיפה לעצים נבחרים אותם

סיבוכיות: o(n)

**שם הפונקציה:**  **public** **static boolean** removePage(GameObject o)

המטודה מוחקת דף מן הדפים

סיבוכיות: o(1)

מבנה Slender

**Private static GameObject \_object //אובייקט הסלנדר**

**Private static Material \_material //סוג החומר והטקסטורה של הסלנדר**

**Private static int \_timer //כל כמה שניות הסלנדר יזוז**

**Private static Random rnd //ראנדום**

**שם הפונקציה:**  **public** **void** init()

המטודה מאתחלת את הסלנדר

סיבוכיות: o(1)

**שם הפונקציה:**  **public** **void** update(float difficulty, Vector3f playerPosition, Quaternion playerRotation)

המטודה מעדכנת את הסלנדר בהתאם לדרגת הקושי והגדרות השחקן

סיבוכיות: o(1)

**3.3 מבנה הנתונים :**

מבנה הנתונים שבו השתמשתי הוא גרף/עץ של אובייקטי המשחק, כאשר לכל אובייקט יש רשימה של מאפייני משחק לעדכון וציור על המסך.

**4. מדריך למשתמש**

**4.1 הוראות התקנה:**

לפני הפעלת התוכנית יש להתקינה במחשב. כלומר צריך להעתיק את התיקיה בשם SLENDER Game למקום כלשהו שנוח לכם על המחשב לאחר מכן צריך להיכנס לתוך תיקיה הזאת וללחוץ על הקובץ SLENDER.jar של התוכנית ומה שנותר זה רק לשחק.

* 1. **הוראות הפעלה:**

אחרי שהתקנתם את המשחק עליכם פשוט לפעיל את הקובץ SLENDER.jar ולהתחיל לשחק לפי ההסברים שהוצגו בפרקים הקודמים.

* 1. **הודעות מערכת :**

ישנן מספר הודעות מערכת:

* + - 1. הודעה על סיום המשחק וניצחון על אסיפת כל הדפים על ידי השחקן
      2. הודעה האם הסלנדר תפס אותך

**5. נספח – קוד מקור**

מחלקת ראשית

**public** **class** Main

{

**public** **static** **void** main(String[]args)

{

CoreEngine engine = **new** CoreEngine(800, 600, 60, **new** FinalGame());

engine.createWindow();

engine.start();

}

}

מחלקת המנוע הראשית

**public** **class** CoreEngine

{

**private** **boolean** \_isRunning; //האם המנוע רץ

**private** Game \_game; //המשחק

**private** **int** \_width; //רוחב המסך ברזולוצייה

**private** **int** \_height; //גובה המסך ברזולוציה

**private** **double** \_frameTime; //הזמן שלוקח לכל פריים להתעדכן

**private** RenderingEngine renderingEngine; //מנוע הציור

**public** CoreEngine(**int** width, **int** height, **double** frameRate, Game game)

{

**this**.\_isRunning = **false**;

**this**.\_width = width;

**this**.\_height = height;

**this**.\_game = game;

**this**.\_frameTime = 1.0/frameRate;

}

**public** **void** createWindow()

{

Window.createWindow(**this**.\_width, **this**.\_height, "Strawberry");

System.***out***.println("OpenGL: "+RenderingEngine.getGlVersion());

**this**.renderingEngine = **new** RenderingEngine();

}

**public** **void** start()

{

**if**(**this**.\_isRunning)

**return**;

run();

}

**public** **void** stop()

{

**if**(!**this**.\_isRunning)

**return**;

**this**.\_isRunning = **false**;

}

**private** **void** run()

{

**this**.\_game.init();

**this**.\_isRunning = **true**;

**int** frames = 0;

**long** frameCounter = 0;

**double** lastTime = Time.*getTime*();

**double** unprocessedTime = 0;

**while**(**this**.\_isRunning)

{

**boolean** render = **false**;

**double** startTime = Time.*getTime*();

**double** passedTime = startTime-lastTime;

lastTime = startTime;

unprocessedTime += passedTime;

frameCounter += passedTime;

**while**(unprocessedTime > \_frameTime)

{

render = **true**;

unprocessedTime -= \_frameTime;

**if**(Window.isCloseRequested())

stop();

**this**.\_game.input((**float**)\_frameTime);

Input.*update*();

**this**.\_game.update((**float**)\_frameTime);

**if**(frameCounter >= 1.0)

{

System.***out***.println(frames);

frames = 0;

frameCounter = 0;

}

}

**if**(render)

{

\_game.render(renderingEngine);

Window.render();

frames++;

}

**else**

{

**try**

{

Thread.*sleep*(1);

}

**catch** (InterruptedException e)

{

e.printStackTrace();

}

}

}

**this**.\_game.destroy();

*cleanUp*();

}

**public** **static** **void** cleanUp()

{

Window.dispose();

System.*exit*(0);

}

}

מחלקת משחק

**public** **abstract** **class** Game

{

**private** GameObject \_root = **new** GameObject(); //הגרף הראשי

**public** **abstract** **void** init();

**public** **void** input(**float** delta)

{

**if**(MenuManager.*gameState* == GameState.***Playing*** && MenuManager.*menuState* == MenuState.***StartGame***)

{

getRootObject().input(delta);

**if**(Input.*getKeyDown*(Keyboard.***KEY\_P*** | Keyboard.***KEY\_ESCAPE***))

{

MenuManager.*gameState* = GameState.***Paused***;

MenuManager.*menuState* = MenuState.***StartGame***;

RenderingEngine.*initializeGL2D*();

}

}

}

**public** **void** update(**float** delta)

{

**if**(MenuManager.*gameState* == GameState.***Playing*** && MenuManager.*menuState* == MenuState.***StartGame***)

{

**if**(!\_root.isImportant())

\_root.setImportant(**true**);

getRootObject().update(delta);

}

MenuManager.*update*();

}

**public** **void** render(RenderingEngine engine)

{

**if**(MenuManager.*gameState* == GameState.***Playing*** && MenuManager.*menuState* == MenuState.***StartGame***)

engine.render(getRootObject());

MenuManager.*render*();

}

**public** **void** destroy()

{

\_root = **null**;

}

**public** **void** addToRoot(GameObject object)

{

getRootObject().addChild(object);

}

**private** GameObject getRootObject() { **return** **this**.\_root; }

}

מחלקת אובייקט משחק

**public** **class** GameObject

{

**private** ArrayList<GameObject> \_children;//לכל אובייקט יש בנים

**private** ArrayList<GameComponent> \_components;//לכל אובייקט תהיה רשימה של הגדרות

**private** Transform \_transform;//לכל אובייקט יהיו ההגדרות הפיזיקליות שלו (סיבוב, מיקום..)

**private** **boolean** \_important; //אם חשוב לצייר אותו בכל פריים

**public** GameObject()

{

**this**.\_children = **new** ArrayList<GameObject>();

**this**.\_components = **new** ArrayList<GameComponent>();

**this**.\_transform = **new** Transform();

\_transform.setScale(**new** Vector3f(0.5f, 0.5f, 0.5f));

setImportant(**false**);

}

**public** GameObject addChild(GameObject o)

{

**this**.\_children.add(o);

o.getTransform().setParent(\_transform);

**return** **this**;

}

**public** GameObject addComponent(GameComponent c)

{

**this**.\_components.add(c);

c.setParent(**this**);

**return** **this**;

}

**public** **void** input(**float** delta)

{

\_transform.update();

**for**(GameComponent component : **this**.\_components)

component.input(delta);

**for**(GameObject child : **this**.\_children)

child.input(delta);

}

**public** **void** update(**float** delta)

{

**for**(GameComponent component : **this**.\_components)

component.update(delta);

**for**(GameObject child : **this**.\_children)

child.update(delta);

}

**public** **void** render(Shader shader, RenderingEngine engine, Vector3f position)

{

**int** range = 300;

**if**(isImportant() || (

(position.getX() + range > getTransform().getPosition().getX() &&

position.getX() - range < getTransform().getPosition().getX()) ||

(position.getZ() + range > getTransform().getPosition().getZ() &&

position.getZ() - range < getTransform().getPosition().getZ())))

{

**for**(GameComponent component : **this**.\_components)

component.render(shader, engine);

**for**(GameObject child : **this**.\_children)

child.render(shader, engine, position);

}

}

**public** **void** addToRenderingEngine(RenderingEngine engine)

{

**for**(GameComponent component : **this**.\_components)

component.addToRenderingEngine(engine);

**for**(GameObject child : **this**.\_children)

child.addToRenderingEngine(engine);

}

**public** Transform getTransform() { **return** **this**.\_transform; }

**public** **boolean** isImportant() { **return** \_important; }

**public** **void** setImportant(**boolean** important) { **this**.\_important = important; }

}

מחלקת קלט

**public** **class** Input

{

**public** **static** **final** **int** ***NUM\_KEYCODES*** = 256; //מקשי המקלדת

**public** **static** **final** **int** ***NUM\_MOUSEBUTTONS*** = 5; //מקשי העכבר

**private** **static** **boolean**[] *\_lastKeys* = **new** **boolean**[***NUM\_KEYCODES***]; //המקשים מהפריים הקודם

**private** **static** **boolean**[] *\_lastMouse* = **new** **boolean**[***NUM\_MOUSEBUTTONS***]; //מקשי העכבר מהפריים הקודם

**public** **static** **void** update()

{

**for** (**int** i = 0; i < ***NUM\_KEYCODES***; i++)

{

*\_lastKeys*[i] = *getKey*(i);

}

**for** (**int** i = 0; i < ***NUM\_MOUSEBUTTONS***; i++)

{

*\_lastMouse*[i] = *getMouse*(i);

}

}

**public** **static** **boolean** getKey(**int** keyCode)

{

**return** Keyboard.*isKeyDown*(keyCode);

}

**public** **static** **boolean** getKeyDown(**int** keyCode)

{

**return** *getKey*(keyCode) && !*\_lastKeys*[keyCode];

}

**public** **static** **boolean** getKeyUp(**int** keyCode)

{

**return** !*getKey*(keyCode) && *\_lastKeys*[keyCode];

}

**public** **static** **boolean** getMouse(**int** mouseButton)

{

**return** Mouse.*isButtonDown*(mouseButton);

}

**public** **static** **boolean** getMouseDown(**int** mouseButton)

{

**return** *getMouse*(mouseButton) && !*\_lastMouse*[mouseButton];

}

**public** **static** **boolean** getMouseUp(**int** mouseButton)

{

**return** !*getMouse*(mouseButton) && *\_lastMouse*[mouseButton];

}

**public** **static** Vector2f getMousePos()

{

**return** **new** Vector2f(Mouse.*getX*(), Mouse.*getY*());

}

**public** **static** **void** setMousePos(Vector2f position)

{

Mouse.*setCursorPosition*((**int**)position.getX(), (**int**)position.getY());

}

**public** **static** **void** setCursor(**boolean** enabled)

{

Mouse.*setGrabbed*(!enabled);

}

}

מחלקת המטריקס

**public** **class** Matrix4f

{

**private** **float**[][] \_m;

**public** Matrix4f()

{

**this**.\_m = **new** **float**[4][4];

}

**public** Matrix4f initIdentity()

{

**this**.\_m[0][0] = 1; **this**.\_m[0][1] = 0; **this**.\_m[0][2] = 0; **this**.\_m[0][3] = 0;

**this**.\_m[1][0] = 0; **this**.\_m[1][1] = 1; **this**.\_m[1][2] = 0; **this**.\_m[1][3] = 0;

**this**.\_m[2][0] = 0; **this**.\_m[2][1] = 0; **this**.\_m[2][2] = 1; **this**.\_m[2][3] = 0;

**this**.\_m[3][0] = 0; **this**.\_m[3][1] = 0; **this**.\_m[3][2] = 0; **this**.\_m[3][3] = 1;

**return** **this**;

}

**public** Matrix4f initTranslation(**float** x, **float** y, **float** z)

{

**this**.\_m[0][0] = 1; **this**.\_m[0][1] = 0; **this**.\_m[0][2] = 0; **this**.\_m[0][3] = x;

**this**.\_m[1][0] = 0; **this**.\_m[1][1] = 1; **this**.\_m[1][2] = 0; **this**.\_m[1][3] = y;

**this**.\_m[2][0] = 0; **this**.\_m[2][1] = 0; **this**.\_m[2][2] = 1; **this**.\_m[2][3] = z;

**this**.\_m[3][0] = 0; **this**.\_m[3][1] = 0; **this**.\_m[3][2] = 0; **this**.\_m[3][3] = 1;

**return** **this**;

}

**public** Matrix4f initPerspective(**float** fov, **float** aspect, **float** zFar, **float** zNear)

{

**float** tanHalfOV = (**float**)Math.*tan*(fov / 2);

**float** zRange = zNear - zFar;

**this**.\_m[0][0] = 1.0f / (aspect \* tanHalfOV);**this**.\_m[0][1] = 0; **this**.\_m[0][2] = 0; **this**.\_m[0][3] = 0;

**this**.\_m[1][0] = 0; **this**.\_m[1][1] = 1.0f / tanHalfOV;**this**.\_m[1][2] = 0; **this**.\_m[1][3] = 0;

**this**.\_m[2][0] = 0; **this**.\_m[2][1] = 0; **this**.\_m[2][2] = (-zNear - zFar) / zRange; **this**.\_m[2][3] = 2 \* zFar \* zNear / zRange;

**this**.\_m[3][0] = 0; **this**.\_m[3][1] = 0; **this**.\_m[3][2] = 1; **this**.\_m[3][3] = 0;

**return** **this**;

}

**public** Matrix4f initOrthographic(**float** left, **float** right, **float** bottom, **float** top, **float** near, **float** far)

{

**float** width = right - left;

**float** height = top - bottom;

**float** depth = far - near;

**this**.\_m[0][0] = 2 / width; **this**.\_m[0][1] = 0; **this**.\_m[0][2] = 0; **this**.\_m[0][3] = -(right + left) / width;

**this**.\_m[1][0] = 0; **this**.\_m[1][1] = 2 / height; **this**.\_m[1][2] = 0; **this**.\_m[1][3] = -(top + bottom) / height;

**this**.\_m[2][0] = 0; **this**.\_m[2][1] = 0; **this**.\_m[2][2] = -2 / depth; **this**.\_m[2][3] = -(far + near) / depth;

**this**.\_m[3][0] = 0; **this**.\_m[3][1] = 0; **this**.\_m[3][2] = 0; **this**.\_m[3][3] = 1;

**return** **this**;

}

**public** Matrix4f initRotation(**float** x, **float** y, **float** z)

{

Matrix4f rx = **new** Matrix4f();

Matrix4f ry = **new** Matrix4f();

Matrix4f rz = **new** Matrix4f();

x = (**float**)Math.*toRadians*(x);

y = (**float**)Math.*toRadians*(y);

z = (**float**)Math.*toRadians*(z);

rx.\_m[0][0] = 1; rx.\_m[0][1] = 0; rx.\_m[0][2] = 0; rx.\_m[0][3] = 0;

rx.\_m[1][0] = 0; rx.\_m[1][1] = (**float**)Math.*cos*(x);rx.\_m[1][2] = -(**float**)Math.*sin*(x);rx.\_m[1][3] = 0;

rx.\_m[2][0] = 0; rx.\_m[2][1] = (**float**)Math.*sin*(x);rx.\_m[2][2] = (**float**)Math.*cos*(x); rx.\_m[2][3] = 0;

rx.\_m[3][0] = 0; rx.\_m[3][1] = 0; rx.\_m[3][2] = 0; rx.\_m[3][3] = 1;

ry.\_m[0][0] = (**float**)Math.*cos*(y);ry.\_m[0][1] = 0; ry.\_m[0][2] = -(**float**)Math.*sin*(y);ry.\_m[0][3] = 0;

ry.\_m[1][0] = 0; ry.\_m[1][1] = 1; ry.\_m[1][2] = 0; ry.\_m[1][3] = 0;

ry.\_m[2][0] = (**float**)Math.*sin*(y);ry.\_m[2][1] = 0; ry.\_m[2][2] = (**float**)Math.*cos*(y); ry.\_m[2][3] = 0;

ry.\_m[3][0] = 0; ry.\_m[3][1] = 0; ry.\_m[3][2] = 0; ry.\_m[3][3] = 1;

//Same as Vector2

rz.\_m[0][0] = (**float**)Math.*cos*(z);rz.\_m[0][1] = -(**float**)Math.*sin*(z);rz.\_m[0][2] = 0; rz.\_m[0][3] = 0;

rz.\_m[1][0] = (**float**)Math.*sin*(z);rz.\_m[1][1] = (**float**)Math.*cos*(z);rz.\_m[1][2] = 0; rz.\_m[1][3] = 0;

rz.\_m[2][0] = 0; rz.\_m[2][1] = 0; rz.\_m[2][2] = 1; rz.\_m[2][3] = 0;

rz.\_m[3][0] = 0; rz.\_m[3][1] = 0; rz.\_m[3][2] = 0; rz.\_m[3][3] = 1;

**this**.\_m = rz.mul(ry.mul(rx)).getMatrixCopy();

**return** **this**;

}

**public** Matrix4f initScale(**float** x, **float** y, **float** z)

{

**this**.\_m[0][0] = x; **this**.\_m[0][1] = 0; **this**.\_m[0][2] = 0; **this**.\_m[0][3] = 0;

**this**.\_m[1][0] = 0; **this**.\_m[1][1] = y; **this**.\_m[1][2] = 0; **this**.\_m[1][3] = 0;

**this**.\_m[2][0] = 0; **this**.\_m[2][1] = 0; **this**.\_m[2][2] = z; **this**.\_m[2][3] = 0;

**this**.\_m[3][0] = 0; **this**.\_m[3][1] = 0; **this**.\_m[3][2] = 0; **this**.\_m[3][3] = 1;

**return** **this**;

}

**public** Matrix4f initRotation(Vector3f forward, Vector3f up)

{

Vector3f f = forward.normalize();

Vector3f r = up.normalize();

r = r.cross(f);

Vector3f u = f.cross(r);

**return** initRotation(f, u, r);

}

**public** Matrix4f initRotation(Vector3f forward, Vector3f up, Vector3f right)

{

Vector3f f = forward;

Vector3f r = right;

Vector3f u = up;

**this**.\_m[0][0] = r.getX(); **this**.\_m[0][1] = r.getY(); **this**.\_m[0][2] = r.getZ(); **this**.\_m[0][3] = 0;

**this**.\_m[1][0] = u.getX(); **this**.\_m[1][1] = u.getY(); **this**.\_m[1][2] = u.getZ(); **this**.\_m[1][3] = 0;

**this**.\_m[2][0] = f.getX(); **this**.\_m[2][1] = f.getY(); **this**.\_m[2][2] = f.getZ(); **this**.\_m[2][3] = 0;

**this**.\_m[3][0] = 0; **this**.\_m[3][1] = 0; **this**.\_m[3][2] = 0; **this**.\_m[3][3] = 1;

**return** **this**;

}

**public** Vector3f transform(Vector3f r)

{

**return** **new** Vector3f(**this**.\_m[0][0] \* r.getX() + **this**.\_m[0][1] \* r.getY() + **this**.\_m[0][2] \* r.getZ() + **this**.\_m[0][3],

**this**.\_m[1][0] \* r.getX() + **this**.\_m[1][1] \* r.getY() + **this**.\_m[1][2] \* r.getZ() + **this**.\_m[1][3],

**this**.\_m[2][0] \* r.getX() + **this**.\_m[2][1] \* r.getY() + **this**.\_m[2][2] \* r.getZ() + **this**.\_m[2][3]);

}

**public** Matrix4f mul(Matrix4f r)

{

Matrix4f res = **new** Matrix4f();

**for**(**int** i = 0; i < 4; i++)

{

**for**(**int** j = 0; j<4; j++)

{

res.set(i, j, **this**.\_m[i][0] \* r.get(0, j) +

**this**.\_m[i][1] \* r.get(1, j) +

**this**.\_m[i][2] \* r.get(2, j) +

**this**.\_m[i][3] \* r.get(3, j));

}

}

**return** res;

}

**public** **float**[][] getMatrixCopy()

{

**float**[][] copy = **new** **float**[4][4];

**for** (**int** i = 0; i < copy.length; i++) {

**for** (**int** j = 0; j < copy.length; j++) {

copy[i][j] = **this**.\_m[i][j];

}

}

**return** copy;

}

**public** **float** get(**int** x, **int** y) { **return** **this**.\_m[x][y]; }

**public** **void** setMatrix(**float**[][] \_m) { **this**.\_m = \_m; }

**public** **void** set(**int** x, **int** y, **float** value) { **this**.\_m[x][y] = value; }

}

מחלקת הקווטרניון (משומש לבדיקת סיבוב)

**public** **class** Quaternion

{

**private** **float** \_x;

**private** **float** \_y;

**private** **float** \_z;

**private** **float** \_w;

**public** Quaternion(**float** x, **float** y, **float** z, **float** w)

{

**this**.\_x = x;

**this**.\_y = y;

**this**.\_z = z;

**this**.\_w = w;

}

**public** Quaternion(Vector3f axis, **float** angle)

{

**float** sinHalfAngle = (**float**) Math.*sin*(angle / 2);

**float** cosHalfAngle = (**float**) Math.*cos*(angle / 2);

**this**.\_x = axis.getX() \* sinHalfAngle;

**this**.\_y = axis.getY() \* sinHalfAngle;

**this**.\_z = axis.getZ() \* sinHalfAngle;

**this**.\_w = cosHalfAngle;

}

**public** **float** length()

{

**return** (**float**)Math.*sqrt*(**this**.\_x \* **this**.\_x + **this**.\_y \* **this**.\_y + **this**.\_z \* **this**.\_z + **this**.\_w \* **this**.\_w);

}

**public** Quaternion normalize()

{

**float** length = length();

**return** **new** Quaternion(**this**.\_x / length, **this**.\_y / length, **this**.\_z / length, **this**.\_w / length);

}

**public** Quaternion conjugate()

{

**return** **new** Quaternion(-**this**.\_x, -**this**.\_y, -**this**.\_z, **this**.\_w);

}

**public** Quaternion mul(**float** r)

{

**return** **new** Quaternion(**this**.\_x \* r, **this**.\_y \* r, **this**.\_z \* r, **this**.\_w \* r);

}

**public** Quaternion mul(Quaternion r)

{

**float** w\_ = **this**.\_w \* r.getW() - **this**.\_x \* r.getX() - **this**.\_y \* r.getY() - **this**.\_z \* r.getZ();

**float** x\_ = **this**.\_x \* r.getW() + **this**.\_w \* r.getX() + **this**.\_y \* r.getZ() - **this**.\_z \* r.getY();

**float** y\_ = **this**.\_y \* r.getW() + **this**.\_w \* r.getY() + **this**.\_z \* r.getX() - **this**.\_x \* r.getZ();

**float** z\_ = **this**.\_z \* r.getW() + **this**.\_w \* r.getZ() + **this**.\_x \* r.getY() - **this**.\_y \* r.getX();

**return** **new** Quaternion(x\_, y\_, z\_, w\_);

}

**public** Quaternion mul(Vector3f r)

{

**float** w\_ = -**this**.\_x \* r.getX() - **this**.\_y \* r.getY() - **this**.\_z \* r.getZ();

**float** x\_ = **this**.\_w \* r.getX() + **this**.\_y \* r.getZ() - **this**.\_z \* r.getY();

**float** y\_ = **this**.\_w \* r.getY() + **this**.\_z \* r.getX() - **this**.\_x \* r.getZ();

**float** z\_ = **this**.\_w \* r.getZ() + **this**.\_x \* r.getY() - **this**.\_y \* r.getX();

**return** **new** Quaternion(x\_, y\_, z\_, w\_);

}

**public** Quaternion lookAt(Vector3f lookAtPos, Vector3f up)

{

Vector3f forward = lookAtPos.getCopy().normalize();

Vector3f right = up.cross(forward);

**float** w\_ = (**float**) (Math.*sqrt*(1.0f + right.getX() + up.getY() + forward.getZ()) \* 0.5f);

**float** recip = 1.0f / (4.0f \* w\_);

**float** x\_ = (forward.getY() - up.getZ()) \* recip;

**float** y\_ = (right.getZ() - forward.getX()) \* recip;

**float** z\_ = (up.getX() - right.getY()) \* recip;

**return** **new** Quaternion(x\_, y\_, z\_, w\_);

}

**public** Matrix4f toRotationMatrix()

{

Vector3f forward = **new** Vector3f(2.0f \* (**this**.\_x \* **this**.\_z - **this**.\_w \* **this**.\_y),

2.0f \* (**this**.\_y \* **this**.\_z + **this**.\_w \* **this**.\_x),

1.0f - 2.0f \* (**this**.\_x \* **this**.\_x + **this**.\_y \* **this**.\_y));

Vector3f up = **new** Vector3f(2.0f \* (**this**.\_x \* **this**.\_y + **this**.\_w \* **this**.\_z),

1.0f - 2.0f \* (**this**.\_x \* **this**.\_x + **this**.\_z \* **this**.\_z),

2.0f \* (**this**.\_y \* **this**.\_z - **this**.\_w \* **this**.\_x));

Vector3f right = **new** Vector3f(1.0f - 2.0f \* (**this**.\_y \* **this**.\_y + **this**.\_z \* **this**.\_z),

2.0f \* (**this**.\_x \* **this**.\_y - **this**.\_w \* **this**.\_z),

2.0f \* (**this**.\_x \* **this**.\_z + **this**.\_w \* **this**.\_y));

**return** **new** Matrix4f().initRotation(forward, up, right);

}

**public** Vector3f getForward() { **return** **new** Vector3f(0,0,1).rotate(**this**); }

**public** Vector3f getBack() { **return** **new** Vector3f(0,0,-1).rotate(**this**); }

**public** Vector3f getUp() { **return** **new** Vector3f(0,1,0).rotate(**this**); }

**public** Vector3f getDown() { **return** **new** Vector3f(0,-1,0).rotate(**this**); }

**public** Vector3f getRight() { **return** **new** Vector3f(1,0,0).rotate(**this**); }

**public** Vector3f getLeft() { **return** **new** Vector3f(-1,0,0).rotate(**this**); }

**public** **float** getX() { **return** \_x; }

**public** **float** getY() { **return** \_y; }

**public** **float** getZ() { **return** \_z; }

**public** **float** getW() { **return** \_w; }

**public** **void** setX(**float** x) { **this**.\_x = x; }

**public** **void** setY(**float** y) { **this**.\_y = y; }

**public** **void** setZ(**float** z) { **this**.\_z = z; }

**public** **void** setW(**float** w) { **this**.\_w = w; }

**public** Quaternion set(Quaternion r) { **this**.\_x = r.\_x; **this**.\_y = r.\_y; **this**.\_z = r.\_z; **this**.\_w = r.\_w; **return** **this**; }

**public** **boolean** equals(Quaternion other)

{

**return** \_x == other.\_x && \_y == other.\_y && \_z == other.\_z && \_w == other.\_w;

}

}

מחלקת הזמן

**public** **class** Time

{

**private** **static** **final** **long** ***SECOND*** = 1000000000l;

**public** **static** **double** getTime()

{

**return** (**double**)System.*nanoTime*() / (**double**)***SECOND***;

}

}

מחלקת השינוי וההגדרות

**public** **class** Transform

{

**private** Transform \_parent; //האבא

**private** Matrix4f \_parentMatrix; //מטריקס האבא

**private** Vector3f \_position; //מיקום בחלל התלת מימדי

**private** Quaternion \_rotation; //סיבוב האובייקט

**private** Vector3f \_scale; //גודל לכל ציר

**private** Vector3f \_oldPosition; //המיקום מהפריים הקודם

**private** Quaternion \_oldRotation; //הסיבוב מהפריים הקודם

**private** Vector3f \_oldScale; //הגודל מהפריים הקודם

**public** Transform()

{

**this**.setPosition(**new** Vector3f(0, 0, 0));

**this**.\_rotation = **new** Quaternion(0,0,0, 1);

**this**.\_scale = **new** Vector3f(1, 1, 1);

**this**.\_parentMatrix = **new** Matrix4f().initIdentity();

**this**.\_oldPosition = Vector3f.*ZERO*;

**this**.\_oldRotation = **new** Quaternion(0,0,0,0);

**this**.\_oldScale = **new** Vector3f(1,1,1);

}

**public** **void** update()

{

**if**(**this**.\_oldPosition != **null**)

{

\_oldPosition = \_position;

\_oldScale = \_scale;

\_oldRotation = \_rotation;

}

**else**

{

**this**.\_oldPosition = Vector3f.*ZERO*.set(**this**.\_position).add(1.0f);

**this**.\_oldRotation = **new** Quaternion(0,0,0,0).set(**this**.\_rotation).mul(0.5f);

**this**.\_oldScale = Vector3f.*ZERO*.set(**this**.\_scale).add(1.0f);

}

}

**public** **void** rotate(Vector3f axis, **float** angle)

{

**this**.\_rotation = **new** Quaternion(axis, angle).mul(**this**.\_rotation).normalize();

}

**public** **boolean** hasChanged()

{

**if**(**this**.\_parent != **null** && **this**.\_parent.hasChanged())

**return** **true**;

**if** (!**this**.\_position.equals(**this**.\_oldPosition)) **return** **true**;

**if** (!**this**.\_scale.equals(**this**.\_oldScale)) **return** **true**;

**if** (!**this**.\_rotation.equals(**this**.\_oldRotation)) **return** **true**;

**return** **false**;

}

**public** Matrix4f getTransformation()

{

Matrix4f translationMatrix = **new** Matrix4f().initTranslation(**this**.\_position.getX(),

**this**.\_position.getY(), **this**.\_position.getZ());

Matrix4f rotationMatrix = **this**.\_rotation.toRotationMatrix

Matrix4f scaleMatrix = **new** Matrix4f().initScale(**this**.\_scale.getX(), **this**.\_scale.getY(), **this**.\_scale.getZ());

**return** getParentMatrix().mul(translationMatrix.mul(rotationMatrix.mul(scaleMatrix)));

}

**private** Matrix4f getParentMatrix()

{

**if**(\_parent != **null** && \_parent.hasChanged())

\_parentMatrix = \_parent.getTransformation();

**return** **this**.\_parentMatrix;

}

**public** Vector3f getTransformedPosition()

{

**return** getParentMatrix().transform(**this**.\_position);

}

**public** Quaternion getTransformedRotation()

{

Quaternion parentRotation = **new** Quaternion(0, 0, 0, 1);

**if**(\_parent != **null**)

parentRotation = **this**.\_parent.getTransformedRotation();

**return** parentRotation.mul(\_rotation);

}

**public** **void** setParent(Transform parent) { **this**.\_parent = parent; }

**public** Quaternion getRotation() { **return** **this**.\_rotation; }

**public** **void** setRotation(Quaternion rotation) { **this**.\_rotation = rotation; }

**public** Vector3f getScale() { **return** **this**.\_scale; }

**public** **void** setScale(Vector3f scale) { **this**.\_scale = scale; }

**public** Vector3f getPosition() { **return** \_position; }

**public** **void** setPosition(Vector3f \_position) { **this**.\_position = \_position; }

}

מחלקת הנקודה הדו מימדית

**public** **class** Vector2f

{

**public** **static** Vector2f *ZERO* = **new** Vector2f();

**private** **float** \_x;

**private** **float** \_y;

**public** Vector2f() {

**this**(0, 0);

}

**public** Vector2f(Vector2f src) {

set(src);

}

**public** Vector2f(**float** x, **float** y) {

set(x, y);

}

**public** **void** set(**float** x, **float** y) {

**this**.\_x = x;

**this**.\_y = y;

}

**public** Vector2f set(Vector2f src) {

**this**.\_x = src.getX();

**this**.\_y = src.getY();

**return** **this**;

}

**public** **float** length()

{

**return** (**float**)Math.*sqrt*(**this**.\_x \* **this**.\_x + **this**.\_y \* **this**.\_y);

}

**public** **float** max()

{

**return** Math.*max*(**this**.\_x, **this**.\_y);

}

**public** Vector2f negate()

{

**this**.\_x = -**this**.\_x;

**this**.\_y = -**this**.\_y;

**return** **this**;

}

**public** Vector2f normalized()

{

**float** l = length();

**return** **new** Vector2f(**this**.\_x / l, **this**.\_y / l);

}

**public** **float** dot(Vector2f r)

{

**return** r.\_x \* **this**.\_x + r.\_y \* **this**.\_y;

}

**public** **float** cross(Vector2f r)

{

**return** **this**.\_x \* r.\_y - **this**.\_y \* r.\_x;

}

**public** Vector2f lerp(Vector2f dest, **float** lerpFactor)

{

**return** dest.sub(**this**).mul(lerpFactor).add(**this**);

}

**public** Vector2f rotate(**float** angle)

{

**double** radians = Math.*toRadians*(angle);

**double** cos = Math.*cos*(radians);

**double** sin = Math.*sin*(radians);

**return** **new** Vector2f((**float**)(**this**.\_x \* cos - **this**.\_y \* sin), (**float**)(**this**.\_x \* sin + **this**.\_y \* cos));

}

**public** Vector2f add(Vector2f r)

{

**return** **new** Vector2f(r.\_x + **this**.\_x, r.\_y + **this**.\_y);

}

**public** Vector2f add(**float** r)

{

**return** **new** Vector2f(r + **this**.\_x, r + **this**.\_y);

}

**public** Vector2f sub(Vector2f r)

{

**return** **new** Vector2f(**this**.\_x - r.\_x, **this**.\_y - r.\_y);

}

**public** Vector2f sub(**float** r)

{

**return** **new** Vector2f(**this**.\_x - r, **this**.\_y - r);

}

**public** Vector2f mul(Vector2f r)

{

**return** **new** Vector2f(**this**.\_x \* r.\_x, **this**.\_y \* r.\_y);

}

**public** Vector2f mul(**float** r)

{

**return** **new** Vector2f(**this**.\_x \* r, **this**.\_y \* r);

}

**public** Vector2f div(Vector2f r)

{

**return** **new** Vector2f(**this**.\_x / r.\_x, **this**.\_y / r.\_y);

}

**public** Vector2f div(**float** r)

{

**return** **new** Vector2f(**this**.\_x / r, **this**.\_y / r);

}

**public** Vector2f abs()

{

**return** **new** Vector2f(Math.*abs*(**this**.\_x), Math.*abs*(**this**.\_y));

}

**public** String toString()

{

StringBuilder sb = **new** StringBuilder(64);

sb.append("Vector2f[");

sb.append(\_x);

sb.append(", ");

sb.append(\_y);

sb.append(']');

**return** sb.toString();

}

/\*\*

\* **@return** x

\*/

**public** **final** **float** getX() { **return** \_x; }

**public** **final** **float** getY() { **return** \_y; }

**public** **final** **void** setX(**float** x) { **this**.\_x = x; }

**public** **final** **void** setY(**float** y) { **this**.\_y = y; }

**public** **boolean** equals(Vector2f other)

{

**return** (\_x == other.\_x && \_y == other.\_y);

}

}

מחלקת הנקודה התלת מימדית

**public** **class** Vector3f

{

**public** **static** Vector3f *ZERO* = **new** Vector3f();

**private** **float** \_x;

**private** **float** \_y;

**private** **float** \_z;

**public** Vector3f()

{

**this**(0, 0, 0);

}

**public** Vector3f(**float** x, **float** y, **float** z)

{

**this**.\_x = x;

**this**.\_y = y;

**this**.\_z = z;

}

**public** Vector3f(Vector3f r)

{

**this**.\_x = r.\_x;

**this**.\_y = r.\_y;

**this**.\_z = r.\_z;

}

**public** **float** length()

{

**return** (**float**)Math.*sqrt*(**this**.\_x \* **this**.\_x + **this**.\_z \* **this**.\_z + **this**.\_y \* **this**.\_y);

}

**public** **float** dot(Vector3f r)

{

**return** **this**.\_x \* r.\_x + **this**.\_y \* r.\_y + **this**.\_z \* r.\_z;

}

**public** Vector3f reflect(Vector3f direction)

{

**return** **this**.sub(direction.mul(**this**.dot(direction) \* 2));

}

**public** Vector3f cross(Vector3f r)

{

**float** tmpX = **this**.\_y \* r.\_z - **this**.\_z \* r.\_y;

**float** tmpY = **this**.\_z \* r.\_x - **this**.\_x \* r.\_z;

**float** tmpZ = **this**.\_x \* r.\_y - **this**.\_y \* r.\_x;

**return** **new** Vector3f(tmpX, tmpY, tmpZ);

}

**public** Vector3f normalize()

{

**float** length = **this**.length();

**this**.\_x /= length;

**this**.\_y /= length;

**this**.\_z /= length;

**return** **this**;

}

**public** Vector3f rotate(**float** angle, Vector3f axis)

{

**float** sinAngle = (**float**)Math.*sin*(-angle);

**float** cosAngle = (**float**)Math.*cos*(-angle);

**return** **this**.cross(axis.mul(sinAngle)).add( //x

(**this**.mul(cosAngle)).add( //z

axis.mul(**this**.dot(axis.mul(1 - cosAngle))))); //y

}

**public** Vector3f rotate(Quaternion rotation)

{

Quaternion conjugate = rotation.conjugate();

Quaternion w = rotation.mul(**this**).mul(conjugate);

**return** **new** Vector3f(w.getX(), w.getY(), w.getZ());

}

**public** Vector3f lerp(Vector3f dest, **float** lerpFactor)

{

**return** dest.sub(**this**).mul(lerpFactor).add(**this**);

}

**public** Vector3f add(Vector3f r)

{

**return** **new** Vector3f(**this**.\_x + r.\_x, **this**.\_y + r.\_y, **this**.\_z + r.\_z);

}

**public** Vector3f add(**float** r)

{

**return** **new** Vector3f(**this**.\_x + r, **this**.\_y + r, **this**.\_z + r);

}

**public** Vector3f sub(Vector3f r)

{

**return** **new** Vector3f(**this**.\_x - r.\_x, **this**.\_y - r.\_y, **this**.\_z - r.\_z);

}

**public** Vector3f sub(**float** r)

{

**return** **new** Vector3f(**this**.\_x - r, **this**.\_y - r, **this**.\_z - r);

}

**public** Vector3f mul(Vector3f r)

{

**return** **new** Vector3f(**this**.\_x \* r.\_x, **this**.\_y \* r.\_y, **this**.\_z \* r.\_z);

}

**public** Vector3f mul(**float** r)

{

**return** **new** Vector3f(**this**.\_x \* r, **this**.\_y \* r, **this**.\_z \* r);

}

**public** Vector3f div(Vector3f r)

{

**return** **new** Vector3f(**this**.\_x / r.\_x, **this**.\_y / r.\_y, **this**.\_z / r.\_z);

}

**public** Vector3f div(**float** r)

{

**return** **new** Vector3f(**this**.\_x / r, **this**.\_y / r, **this**.\_z / r);

}

**public** **float** max()

{

**return** Math.*max*(\_x, Math.*max*(\_y, \_z));

}

**public** Vector3f max(Vector3f r)

{

Vector3f result = **new** Vector3f();

**if**(**this**.getX() > r.getX()) result.\_x = **this**.getX();

**else** result.\_x = r.getX();

**if**(**this**.getY() > r.getY()) result.\_y = **this**.getY();

**else** result.\_y = r.getY();

**if**(**this**.getZ() > r.getZ()) result.\_z = **this**.getZ();

**else** result.\_z = r.getZ();

**return** result;

}

**public** Vector2f getXY() { **return** **new** Vector2f(\_x, \_y); }

**public** Vector2f getYZ() { **return** **new** Vector2f(\_y, \_z); }

**public** Vector2f getZX() { **return** **new** Vector2f(\_z, \_x); }

**public** Vector2f getYX() { **return** **new** Vector2f(\_y, \_x); }

**public** Vector2f getZY() { **return** **new** Vector2f(\_z, \_y); }

**public** Vector2f getXZ() { **return** **new** Vector2f(\_x, \_z); }

**public** **void** set(**float** x, **float** y, **float** z) { **this**.\_x = x; **this**.\_y = y; **this**.\_z = z; }

**public** Vector3f set(Vector3f r) { **this**.\_x = r.\_x; **this**.\_y = r.\_y; **this**.\_z = r.\_z; **return** **this**; }

**public** **float** getX() { **return** \_x; }

**public** **float** getY() { **return** \_y; }

**public** **float** getZ() { **return** \_z; }

**public** **void** setX(**float** x) { **this**.\_x = x; }

**public** **void** setY(**float** y) { **this**.\_y = y; }

**public** **void** setZ(**float** z) { **this**.\_z = z; }

**public** Vector3f getCopy()

{

**return** **new** Vector3f(\_x, \_y, \_z);

}

**public** **boolean** equals(Vector3f other)

{

**return** \_x == other.\_x && \_y == other.\_y && \_z == other.\_z;

}

**public** String toString()

{

**return** "{"+**this**.\_x+", "+**this**.getY()+", "+**this**.getZ()+"}";

}

}

מחלקת ההפחתה בעוצמת התאורה

**public** **class** Attenuation **extends** Vector3f

{

**public** Attenuation(**float** constant, **float** linear, **float** exponent)

{

**super**(constant, linear, exponent);

}

**public** **float** getConstant() { **return** getX(); }

**public** **float** getLinear() { **return** getY(); }

**public** **float** getExponent() { **return** getZ(); }

}

מחלקת התאורה הסביבתית/ראשית

**public** **class** ForwardAmbient **extends** Shader

{

**private** **static** **final** ForwardAmbient ***instance*** = **new** ForwardAmbient();

**public** **static** ForwardAmbient getInstance()

{

**return** ***instance***;

}

**public** ForwardAmbient()

{

**super**();

addVertexShader(Shader.*loadShader*("forward-ambient.vs"));

addFragmentShader(Shader.*loadShader*("forward-ambient.fs"));

setAttribLocation("position", 0);

setAttribLocation("textCoord", 1);

compileShader();

addUniform("MVP");

addUniform("ambientIntensity");

}

**public** **void** updateUniforms(Transform transform, Material material, RenderingEngine engine)

{

Matrix4f worldMatrix = transform.getTransformation();

Matrix4f projectedMatrix = engine.getMainCamera().getViewProjection().mul(worldMatrix);

material.getTexture("diffuse").bind();

setUniformM("MVP", projectedMatrix);

setUniformV("ambientIntensity", engine.getAmbientLight());

}

}

מחלקת התאורה הישירה

**public** **class** ForwardDirectional **extends** Shader

{

**private** **static** **final** ForwardDirectional ***instance*** = **new** ForwardDirectional();

**public** **static** ForwardDirectional getInstance()

{

**return** ***instance***;

}

**public** ForwardDirectional()

{

**super**();

addVertexShader(Shader.*loadShader*("forward-directional.vs"));

addFragmentShader(Shader.*loadShader*("forward-directional.fs"));

setAttribLocation("position", 0);

setAttribLocation("textCoord", 1);

setAttribLocation("normal", 2);

compileShader();

addUniform("model");

addUniform("MVP");

addUniform("specularIntensity");

addUniform("specularPower");

addUniform("eyePos");

addUniform("directionalLight.base.color");

addUniform("directionalLight.base.intensity");

addUniform("directionalLight.direction");

}

**public** **void** updateUniforms(Transform transform, Material material, RenderingEngine engine)

{

Matrix4f worldMatrix = transform.getTransformation();

Matrix4f projectedMatrix = engine.getMainCamera().getViewProjection().mul(worldMatrix);

material.getTexture("diffuse").bind();

setUniformM("MVP", projectedMatrix);

setUniformM("model", worldMatrix);

setUniformf("specularIntensity", material.getFloat("specularIntensity"));

setUniformf("specularPower", material.getFloat("specularPower"));

setUniformV("eyePos", engine.getMainCamera().getTransform().getTransformedPosition());

setUniform("directionalLight", (DirectionalLight)engine.getActiveLight());

}

**public** **void** setUniform(String uniformName, BaseLight baseLight)

{

setUniformV(uniformName + ".color", baseLight.getColor());

setUniformf(uniformName + ".intensity", baseLight.getIntensity());

}

**public** **void** setUniform(String uniformName, DirectionalLight directionalLight)

{

setUniform(uniformName + ".base", (BaseLight)directionalLight);

setUniformV(uniformName + ".direction", directionalLight.getDirection());

}

}

מחלקת התאורה כנקודה

**public** **class** ForwardPoint **extends** Shader

{

**private** **static** **final** ForwardPoint ***instance*** = **new** ForwardPoint();

**public** **static** ForwardPoint getInstance()

{

**return** ***instance***;

}

**public** ForwardPoint()

{

**super**();

addVertexShader(Shader.*loadShader*("forward-point.vs"));

addFragmentShader(Shader.*loadShader*("forward-point.fs"));

setAttribLocation("position", 0);

setAttribLocation("textCoord", 1);

setAttribLocation("normal", 2);

compileShader();

addUniform("model");

addUniform("MVP");

addUniform("specularIntensity");

addUniform("specularPower");

addUniform("eyePos");

addUniform("pointLight.base.color");

addUniform("pointLight.base.intensity");

addUniform("pointLight.atten.constant");

addUniform("pointLight.atten.linear");

addUniform("pointLight.atten.exponent");

addUniform("pointLight.position");

addUniform("pointLight.range");

}

**public** **void** updateUniforms(Transform transform, Material material, RenderingEngine engine)

{

Matrix4f worldMatrix = transform.getTransformation();

Matrix4f projectedMatrix = engine.getMainCamera().getViewProjection().mul(worldMatrix);

material.getTexture("diffuse").bind();

setUniformM("MVP", projectedMatrix);

setUniformM("model", worldMatrix);

setUniformf("specularIntensity", material.getFloat("specularIntensity"));

setUniformf("specularPower", material.getFloat("specularPower"));

setUniformV("eyePos", engine.getMainCamera().getTransform().getTransformedPosition());

setUniform("pointLight", (PointLight)engine.getActiveLight());

}

**public** **void** setUniform(String uniformName, BaseLight baseLight)

{

setUniformV(uniformName + ".color", baseLight.getColor());

setUniformf(uniformName + ".intensity", baseLight.getIntensity());

}

**private** **void** setUniform(String uniformName, PointLight pointLight)

{

setUniform(uniformName + ".base", (BaseLight)pointLight);

setUniformf(uniformName + ".atten.constant", pointLight.getConstant());

setUniformf(uniformName + ".atten.linear", pointLight.getLinear());

setUniformf(uniformName + ".atten.exponent", pointLight.getExponent());

setUniformV(uniformName + ".position", pointLight.getTransform().getTransformedPosition());

setUniformf(uniformName + ".range", pointLight.getRange());

}

}

מחלקת התאורה הנקודתית (המשודרגת)

**public** **class** ForwardSpot **extends** Shader

{

**private** **static** **final** ForwardSpot ***instance*** = **new** ForwardSpot();

**public** **static** ForwardSpot getInstance()

{

**return** ***instance***;

}

**public** ForwardSpot()

{

**super**();

addVertexShader(Shader.*loadShader*("forward-spot.vs"));

addFragmentShader(Shader.*loadShader*("forward-spot.fs"));

setAttribLocation("position", 0);

setAttribLocation("textCoord", 1);

setAttribLocation("normal", 2);

compileShader();

addUniform("model");

addUniform("MVP");

addUniform("specularIntensity");

addUniform("specularPower");

addUniform("eyePos");

addUniform("spotLight.pointLight.base.color");

addUniform("spotLight.pointLight.base.intensity");

addUniform("spotLight.pointLight.atten.constant");

addUniform("spotLight.pointLight.atten.linear");

addUniform("spotLight.pointLight.atten.exponent");

addUniform("spotLight.pointLight.position");

addUniform("spotLight.pointLight.range");

addUniform("spotLight.direction");

addUniform("spotLight.cutoff");

}

**public** **void** updateUniforms(Transform transform, Material material, RenderingEngine engine)

{

Matrix4f worldMatrix = transform.getTransformation();

Matrix4f projectedMatrix = engine.getMainCamera().getViewProjection().mul(worldMatrix);

material.getTexture("diffuse").bind();

setUniformM("MVP", projectedMatrix);

setUniformM("model", worldMatrix);

setUniformf("specularIntensity", material.getFloat("specularIntensity"));

setUniformf("specularPower", material.getFloat("specularPower"));

setUniformV("eyePos", engine.getMainCamera().getTransform().getTransformedPosition());

setUniform("spotLight", (SpotLight)engine.getActiveLight());

}

**public** **void** setUniform(String uniformName, BaseLight baseLight)

{

setUniformV(uniformName + ".color", baseLight.getColor());

setUniformf(uniformName + ".intensity", baseLight.getIntensity());

}

**private** **void** setUniform(String uniformName, PointLight pointLight)

{

setUniform(uniformName + ".base", (BaseLight)pointLight);

setUniformf(uniformName + ".atten.constant", pointLight.getConstant());

setUniformf(uniformName + ".atten.linear", pointLight.getLinear());

setUniformf(uniformName + ".atten.exponent", pointLight.getExponent());

setUniformV(uniformName + ".position", pointLight.getTransform().getTransformedPosition());

setUniformf(uniformName + ".range", pointLight.getRange());

}

**private** **void** setUniform(String uniformName, SpotLight spotLight)

{

setUniform(uniformName + ".pointLight", (PointLight)spotLight);

setUniformV(uniformName + ".direction", spotLight.getDirection());

setUniformf(uniformName + ".cutoff", spotLight.getCutoff());

}

}

מחלקת החומר

**public** **class** Material

{

**private** HashMap<String, Texture> \_textureHash;

**private** HashMap<String, Vector3f> \_vectorHash;

**private** HashMap<String, Float> \_floatHash;

**public** Material()

{

**this**.\_textureHash = **new** HashMap<String, Texture>();

**this**.\_vectorHash = **new** HashMap<String, Vector3f>();

**this**.\_floatHash = **new** HashMap<String, Float>();

}

**public** **void** addTexture(String name, Texture texture) { **this**.\_textureHash.put(name, texture); }

**public** Texture getTexture(String name)

{

Texture t = **this**.\_textureHash.get(name);

**if**(t != **null**) **return** t;

**return** Texture.*loadTextureID*("java.jpg");

}

**public** **void** addVector(String name, Vector3f vector) { **this**.\_vectorHash.put(name, vector); }

**public** Vector3f getVector(String name)

{

Vector3f v = **this**.\_vectorHash.get(name);

**if**(v != **null**) **return** v;

**return** Vector3f.*ZERO*;

}

**public** **void** addFloat(String name, Float f) { **this**.\_floatHash.put(name, f); }

**public** **float** getFloat(String name)

{

Float f = **this**.\_floatHash.get(name);

**if**(f != **null**) **return** f;

**return** 0;

}

}

מחלקת המודל

**public** **class** Mesh

{

**private** **int** \_size;

**private** **int** \_ibo;

**private** **int** \_vbo;

**public** Mesh(Vertex[] vertices, **int**[] indices)

{

**this**.\_vbo = *glGenBuffers*();

**this**.\_ibo = *glGenBuffers*();

**this**.\_size = 0;

addVertices(vertices, indices, **true**);

}

**public** Mesh()

{

**this**.\_vbo = *glGenBuffers*();

**this**.\_ibo = *glGenBuffers*();

**this**.\_size = 0;

}

**public** Mesh(String filename)

{

**this**.\_vbo = *glGenBuffers*();

**this**.\_ibo = *glGenBuffers*();

**this**.\_size = 0;

loadMesh(filename);

}

**public** **void** addVertices(Vertex[] vertices, **int**[] indices)

{

addVertices(vertices, indices, **false**);

}

**public** **void** addVertices(Vertex[] vertices, **int**[] indices, **boolean** calcNormals)

{

**if**(calcNormals)

calcNormals(vertices, indices);

**this**.\_size = indices.length;

*glBindBuffer*(***GL\_ARRAY\_BUFFER***, **this**.\_vbo);

*glBufferData*(***GL\_ARRAY\_BUFFER***, *createFlippedBuffer*(vertices), ***GL\_STATIC\_DRAW***);

*glBindBuffer*(***GL\_ELEMENT\_ARRAY\_BUFFER***, **this**.\_ibo);

*glBufferData*(***GL\_ELEMENT\_ARRAY\_BUFFER***, *createFlippedBuffer*(indices), ***GL\_STATIC\_DRAW***);

}

**public** **void** render()

{

*glEnableVertexAttribArray*(0);

*glEnableVertexAttribArray*(1);

*glEnableVertexAttribArray*(2);

*glBindBuffer*(***GL\_ARRAY\_BUFFER***, **this**.\_vbo);

*glVertexAttribPointer*(0, 3, ***GL\_FLOAT***, **false**, Vertex.***SIZE*** \* 4, 0);

*glVertexAttribPointer*(1, 2, ***GL\_FLOAT***, **false**, Vertex.***SIZE*** \* 4, 12);

*glVertexAttribPointer*(2, 3, ***GL\_FLOAT***, **false**, Vertex.***SIZE*** \* 4, 20);

*glBindBuffer*(***GL\_ELEMENT\_ARRAY\_BUFFER***, **this**.\_ibo);

*glDrawElements*(***GL\_TRIANGLES***, **this**.\_size, ***GL\_UNSIGNED\_INT***, 0);

//glDrawArrays(GL\_TRIANGLES, 0, this.\_size);

*glDisableVertexAttribArray*(0);

*glDisableVertexAttribArray*(1);

*glDisableVertexAttribArray*(2);

}

**private** **void** calcNormals(Vertex[] vertices, **int**[] indices)

{

**for**(**int** i=0; i<indices.length; i+= 3)

{

**int** i0 = indices[i];

**int** i1 = indices[i+1];

**int** i2 = indices[i+2];

Vector3f v1 = vertices[i1].getPosition().sub(vertices[i0].getPosition());

Vector3f v2 = vertices[i2].getPosition().sub(vertices[i0].getPosition());

Vector3f normal = v1.cross(v2).normalize();

vertices[i0].setNormal(vertices[i0].getNormal().add(normal));

vertices[i1].setNormal(vertices[i1].getNormal().add(normal));

vertices[i2].setNormal(vertices[i2].getNormal().add(normal));

}

**for**(**int** i = 0; i< vertices.length; i++)

vertices[i].setNormal(vertices[i].getNormal().normalize());

}

**private** **static** FloatBuffer createFloatBuffer(**int** size)

{

**return** BufferUtils.*createFloatBuffer*(size);

}

**private** **static** IntBuffer createIntBuffer(**int** size)

{

**return** BufferUtils.*createIntBuffer*(size);

}

**private** **static** IntBuffer createFlippedBuffer(**int**... values)

{

IntBuffer buffer = *createIntBuffer*(values.length);

buffer.put(values);

buffer.flip();

**return** buffer;

}

**private** **static** FloatBuffer createFlippedBuffer(Vertex[] vertices)

{

FloatBuffer buffer = *createFloatBuffer*(vertices.length \* Vertex.***SIZE*** );

**for**(**int** i=0;i<vertices.length; i++)

{

buffer.put(vertices[i].getPosition().getX());

buffer.put(vertices[i].getPosition().getY());

buffer.put(vertices[i].getPosition().getZ());

buffer.put(vertices[i].getTextCoordinates().getX());

buffer.put(vertices[i].getTextCoordinates().getY());

buffer.put(vertices[i].getNormal().getX());

buffer.put(vertices[i].getNormal().getY());

buffer.put(vertices[i].getNormal().getZ());

}

buffer.flip();

**return** buffer;

}

**public** **static** FloatBuffer createFlippedBuffer(Matrix4f value)

{

FloatBuffer buffer = *createFloatBuffer*(4\*4);

**for**(**int** i=0;i<4; i++)

{

**for**(**int** j=0;j<4; j++)

{

buffer.put(value.get(i, j));

}

}

buffer.flip();

**return** buffer;

}

**public** Mesh loadMesh(String filename)

{

String[] splitArr = filename.split("\\.");

String ext = splitArr[splitArr.length - 1];

**if**(!ext.equals("obj"))

{

System.***err***.println("At this point only .obj is supported");

System.*exit*(1);

}

OBJModel test = **new** OBJModel("./res/models/"+filename);

IndexedModel model = test.toIndexedModel();

ArrayList<Vertex> vertices = **new** ArrayList<Vertex>();

**for** (**int** i = 0; i < model.getPositions().size(); i++)

{

vertices.add(**new** Vertex(model.getPositions().get(i),

model.getTextureCoordinates().get(i),

model.getNormals().get(i)));

}

Vertex[] vertexData = **new** Vertex[vertices.size()];

vertices.toArray(vertexData);

Integer[] indexData = **new** Integer[model.getIndices().size()];

model.getIndices().toArray(indexData);

model.calcNormals();

**this**.addVertices(vertexData, Statics.*convertToIntArray*(indexData), **false**);

**return** **null**;

}

}

מחלקת המנוע ציור

**public** **class** RenderingEngine

{

**private** Camera \_mainCamera; //המצלמה הראשית

**private** Vector3f \_ambientLight; //הצבע שיאפיל על כולם

**private** ArrayList<BaseLight> \_lights; //רשימת התאורות לציור

**private** BaseLight \_activeLight; //התאורה שבדיוק מצוירת

**public** RenderingEngine()

{

\_lights = **new** ArrayList<BaseLight>();

\_ambientLight = **new** Vector3f(0.2f, 0.2f, 0.2f);

*glClearColor*(0f, 0f, 0f, 0f);

*initializeGL2D*();

}

**public** **void** render(GameObject object)

{

*clrScreen*();

**this**.\_lights.clear();

object.addToRenderingEngine(**this**);

Shader forwardAmbient = ForwardAmbient.*getInstance*();

object.render(forwardAmbient, **this**, \_mainCamera.getTransform().getPosition());

*glEnable*(***GL\_BLEND***);

*glBlendFunc*(***GL\_ONE***, ***GL\_ONE***);

*glDepthMask*(**false**);

*glDepthFunc*(***GL\_EQUAL***);

{

**for**(BaseLight light : \_lights)

{

\_activeLight = light;

object.render(light.getShader(), **this**, \_mainCamera.getTransform().getPosition());

}

}

*glDepthFunc*(***GL\_LESS***);

*glDepthMask*(**true**);

*glDisable*(***GL\_BLEND***);

**if**(MenuManager.*gameState* == GameState.***Paused***)

**this**.\_mainCamera.\_projection = **new** Matrix4f().initOrthographic(0, Window.*getWidth*() , Window.*getHeight*(), 0, -1, 1);

}

**public** **static** **void** clrScreen()

{

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

//if(!Slender.isInRnage())glClear(GL\_DEPTH\_BUFFER\_BIT);

}

**public** **void** addLight(BaseLight light)

{

**this**.\_lights.add(light);

}

**public** **void** addCamera(Camera camera)

{

setMainCamera(camera);

}

**public** BaseLight getActiveLight()

{

**return** \_activeLight;

}

**public** **static** **void** initializeGL2D()

{

*glPopAttrib*();

*glPushAttrib*(***GL\_ENABLE\_BIT***);

*glViewport*(0, 0, Window.*getWidth*(), Window.*getHeight*());

*glMatrixMode*(***GL\_PROJECTION***);

*glLoadIdentity*();

*glOrtho*(0, Window.*getWidth*() , Window.*getHeight*(), 0, -1, 1);

*glMatrixMode*(***GL\_MODELVIEW***);

*glLoadIdentity*();

}

**public** **static** **void** initializeGL3D()

{

*glPopAttrib*();

*glPushAttrib*(***GL\_ENABLE\_BIT***);

*glEnable*(***GL\_DEPTH\_TEST***);

*glDepthFunc*(***GL\_LEQUAL***);

*glViewport*(0, 0, Window.*getWidth*(), Window.*getHeight*());

*glMatrixMode*(***GL\_PROJECTION***);

*glLoadIdentity*();

*glMatrixMode*(***GL\_MODELVIEW***);

*glLoadIdentity*();

}

**public** Vector3f getAmbientLight() { **return** \_ambientLight; }

**public** **static** String getGlVersion() { **return** *glGetString*(***GL\_VERSION***); }

**public** Camera getMainCamera() { **return** \_mainCamera; }

**public** **void** setMainCamera(Camera mainCamera) { **this**.\_mainCamera = mainCamera; }

}

מחלקת האפקט למודל

**public** **abstract** **class** Shader

{

**private** **int** \_program;

**private** HashMap<String, Integer> \_uniforms;

**public** Shader()

{

\_program = *glCreateProgram*();

**if**(\_program == 0)

{

System.***err***.println("Shader creation has failed: Could not find memory location in constructor");

System.*exit*(1);

}

**this**.\_uniforms = **new** HashMap<String, Integer>();

}

**public** **void** addVertexShader(String text)

{

addProgram(text, ***GL\_VERTEX\_SHADER***);

}

**public** **void** addUniform(String uniform)

{

**int** uniformLocation = *glGetUniformLocation*(\_program, uniform);

//glBindAttribLocation(\_program, 0, uniform);

**if**(uniformLocation == 0xFFFFFFFF)

{

System.***err***.println("Could not find uniform: "+ uniform);

System.*exit*(1);

}

//glBindAttribLocation(\_program, 0, uniform);

**this**.\_uniforms.put(uniform, uniformLocation);

}

**public** **void** addGeometryShader(String text)

{

addProgram(text, ***GL\_GEOMETRY\_SHADER***);

}

**public** **void** addFragmentShader(String text)

{

addProgram(text, ***GL\_FRAGMENT\_SHADER***);

}

**public** **void** bind()

{

*glUseProgram*(\_program);

*glBindAttribLocation*(\_program, 0, "location");

}

**public** **abstract** **void** updateUniforms(Transform transform, Material material, RenderingEngine engine);

**public** **void** setAttribLocation(String attributeName, **int** location)

{

*glBindAttribLocation*(\_program, location, attributeName);

}

**public** **void** compileShader()

{

*glLinkProgram*(\_program);

**if**(*glGetProgrami*(\_program, ***GL\_LINK\_STATUS***) == 0)

{

System.***err***.println(*glGetProgramInfoLog*(\_program, 1024));

System.*exit*(1);

}

*glValidateProgram*(\_program);

**if**(*glGetProgrami*(\_program, ***GL\_VALIDATE\_STATUS***) == 0)

{

System.***err***.println(*glGetProgramInfoLog*(\_program, 1024));

System.*exit*(1);

}

}

**private** **void** addProgram(String text, **int** type)

{

**int** shader = *glCreateShader*(type);

**if**(shader == 0)

{

System.***err***.println("Shader creation has failed: Could not find memory location when adding shader");

System.*exit*(1);

}

*glShaderSource*(shader, text);

*glCompileShader*(shader);

**if**(*glGetShaderi*(shader, ***GL\_COMPILE\_STATUS***) == 0)

{

System.***err***.println(*glGetShaderInfoLog*(shader, 1024));

System.*exit*(1);

}

*glAttachShader*(\_program, shader);

}

**public** **void** setUniformi(String uniformName, **int** value)

{

*glUniform1i*(**this**.\_uniforms.get(uniformName), value);

}

**public** **void** setUniformf(String uniformName, **float** value)

{

*glUniform1f*(**this**.\_uniforms.get(uniformName), value);

}

**public** **void** setUniformV(String uniformName, Vector3f value)

{

*glUniform3f*(**this**.\_uniforms.get(uniformName), value.getX(), value.getY(), value.getZ());

}

**public** **void** setUniformM(String uniformName, Matrix4f value)

{

*glUniformMatrix4*(**this**.\_uniforms.get(uniformName), **true**, Mesh.*createFlippedBuffer*(value));

}

**public** **static** String loadShader(String filename)

{

StringBuilder shaderSource = **new** StringBuilder();

BufferedReader shaderReader = **null**;

**try**

{

shaderReader = **new** BufferedReader(**new** FileReader("./res/shaders/"+filename));

String line;

**while** ((line = shaderReader.readLine()) != **null**)

{

shaderSource.append(line).append("\n");

}

shaderReader.close();

}**catch**(Exception e)

{

e.printStackTrace();

System.*exit*(1);

}

**return** shaderSource.toString();

}

}

מחלקת הטקסטורה

**public** **class** Texture

{

**private** **int** \_id;

**public** Texture(**int** id)

{

**this**.\_id = id;

}

**public** **void** bind()

{

*glBindTexture*(***GL\_TEXTURE\_2D***, **this**.\_id);

}

**public** **int** getId() {

**return** \_id;

}

**public** **void** setId(**int** id) {

**this**.\_id = id;

}

**public** **static** com.strawberry.engine.rendering.Texture loadTextureID(String filename)

{

String[] splitArray = filename.split("\\.");

String ext = splitArray[splitArray.length - 1];

**try**

{

**int** id = TextureLoader.*getTexture*(ext, **new** FileInputStream(**new** File("./res/textures/" + filename))).getTextureID();

**return** **new** com.strawberry.engine.rendering.Texture(id);

}

**catch**(Exception e)

{

e.printStackTrace();

System.*exit*(1);

}

**return** **null**;

}

}

המחלקת הנקודה במודל

**public** **class** Vertex

{

**public** **static** **final** **int** ***SIZE*** = 8;

**private** Vector3f \_position;

**private** Vector2f \_textCoordinates;

**private** Vector3f \_normal;

**public** Vertex(Vector3f pos)

{

**this**(pos, **new** Vector2f(0,0));

}

**public** Vertex(Vector3f pos, Vector2f text)

{

**this**(pos,text, Vector3f.*ZERO*);

}

**public** Vertex(Vector3f pos, Vector2f text, Vector3f normal)

{

**this**.\_position = pos;

**this**.\_textCoordinates = text;

**this**.\_normal = normal;

}

**public** Vector3f getPosition() {

**return** \_position;

}

**public** **void** setPosition(Vector3f position) {

**this**.\_position = position;

}

**public** Vector2f getTextCoordinates() {

**return** \_textCoordinates;

}

**public** **void** setTextCoordinates(Vector2f textCoordinates) {

**this**.\_textCoordinates = textCoordinates;

}

**public** Vector3f getNormal() {

**return** \_normal;

}

**public** **void** setNormal(Vector3f normal) {

**this**.\_normal = normal;

}

}

מחלקת החלון

**public** **class** Window

{

**public** **static** **void** createWindow(**int** width, **int** height, String title)

{

Display.*setTitle*(title);

**try**

{

Display.*setDisplayMode*(**new** DisplayMode(width, height));

Display.*create*();

Keyboard.*create*();

Mouse.*create*();

}

**catch** (LWJGLException e)

{

e.printStackTrace();

}

}

**public** **static** **void** render()

{

Display.*update*();

}

**public** **static** **void** dispose()

{

Display.*destroy*();

Keyboard.*destroy*();

Mouse.*destroy*();

}

**public** **static** **boolean** isCloseRequested()

{

**return** Display.*isCloseRequested*();

}

**public** **static** **int** getWidth()

{

**return** Display.*getDisplayMode*().getWidth();

}

**public** **static** **int** getHeight()

{

**return** Display.*getDisplayMode*().getHeight();

}

**public** Vector2f getCenter()

{

**return** **new** Vector2f(*getWidth*()/2, *getHeight*()/2);

}

}

מחלקת העזר למודל

**public** **class** IndexedModel

{

**private** ArrayList<Vector3f> \_positions;

**private** ArrayList<Vector2f> \_textCoordinates;

**private** ArrayList<Vector3f> \_normals;

**private** ArrayList<Integer> \_indices;

**public** IndexedModel()

{

**this**.\_positions = **new** ArrayList<Vector3f>();

**this**.\_textCoordinates = **new** ArrayList<Vector2f>();

**this**.\_normals = **new** ArrayList<Vector3f>();

**this**.\_indices = **new** ArrayList<Integer>();

}

**public** **void** calcNormals()

{

**for**(**int** i=0; i< \_indices.size(); i+= 3)

{

**int** i0 = \_indices.get(i);

**int** i1 = \_indices.get(i+1);

**int** i2 = \_indices.get(i+2);

Vector3f v1 = \_positions.get(i1).sub(\_positions.get(i0));

Vector3f v2 = \_positions.get(i2).sub(\_positions.get(i0));

Vector3f normal = v1.cross(v2).normalize();

\_normals.get(i0).set(\_normals.get(i0).add(normal));

\_normals.get(i1).set(\_normals.get(i1).add(normal));

\_normals.get(i2).set(\_normals.get(i2).add(normal));

}

**for**(**int** i = 0; i< \_normals.size(); i++)

\_normals.get(i).set(\_normals.get(i).normalize());

}

**public** ArrayList<Vector3f> getPositions() { **return** \_positions; }

**public** ArrayList<Vector2f> getTextureCoordinates() { **return** \_textCoordinates; }

**public** ArrayList<Vector3f> getNormals() { **return** \_normals; }

**public** ArrayList<Integer> getIndices() { **return** \_indices; }

}

מחלקת העזר לקובץ מסוג obj

**public** **class** OBJIndex

{

**public** **int** \_vertexIndex;

**public** **int** \_textureCoordIndex;

**public** **int** \_normalIndex;

**public** **boolean** equlas(Object obj)

{

OBJIndex index = (OBJIndex)obj;

**return** \_vertexIndex == index.\_vertexIndex &&

\_textureCoordIndex == index.\_textureCoordIndex &&

\_normalIndex == index.\_normalIndex;

}

@Override

**public** **int** hashCode()

{

**final** **int** BASE = 17;

**final** **int** MULTIPLIER = 31;

**int** res = BASE;

res = MULTIPLIER \* res + \_vertexIndex;

res = MULTIPLIER \* res + \_textureCoordIndex;

res = MULTIPLIER \* res + \_normalIndex;

**return** res;

}

}

מחלקת המודל מסוג obj

**public** **class** OBJModel

{

**private** ArrayList<Vector3f> \_positions;

**private** ArrayList<Vector2f> \_textCoordinates;

**private** ArrayList<Vector3f> \_normals;

**private** ArrayList<OBJIndex> \_indices;

**private** **boolean** \_hasTextureCoordinates;

**private** **boolean** \_hasNormals;

**public** OBJModel(String filename)

{

**this**.\_positions = **new** ArrayList<Vector3f>();

**this**.\_textCoordinates = **new** ArrayList<Vector2f>();

**this**.\_normals = **new** ArrayList<Vector3f>();

**this**.\_indices = **new** ArrayList<OBJIndex>();

**this**.\_hasTextureCoordinates = **false**;

**this**.\_hasNormals = **false**;

load(filename);

}

**public** **void** load(String filename)

{

BufferedReader reader = **null**;

**try**

{

reader = **new** BufferedReader(**new** FileReader(filename));

String line;

**while** ((line = reader.readLine()) != **null**)

{

String[] tokens = line.split(" ");

tokens = Statics.*removeEmptyStrings*(tokens);

**if**(tokens.length == 0 || tokens[0].equals("#"))

**continue**;

**else** **if**(tokens[0].equals("v"))

{

\_positions.add(**new** Vector3f(Float.*valueOf*(tokens[1]),

Float.*valueOf*(tokens[2]),

Float.*valueOf*(tokens[3])));

}

**else** **if**(tokens[0].equals("vt"))

{

\_textCoordinates.add(**new** Vector2f(Float.*valueOf*(tokens[1]),

Float.*valueOf*(tokens[2])));

}

**else** **if**(tokens[0].equals("vn"))

{

\_normals.add(**new** Vector3f(Float.*valueOf*(tokens[1]),

Float.*valueOf*(tokens[2]),

Float.*valueOf*(tokens[3])));

}

**else** **if**(tokens[0].equals("f"))

{

**for** (**int** i = 0; i < tokens.length - 3; i++)

{

**this**.\_indices.add(parseOBJIndex(tokens[1]));

**this**.\_indices.add(parseOBJIndex(tokens[2 + i]));

**this**.\_indices.add(parseOBJIndex(tokens[3 + i]));

}

}

}

reader.close();

}

**catch**(Exception e)

{

e.printStackTrace();

System.*exit*(1);

}

}

**public** IndexedModel toIndexedModel()

{

IndexedModel res = **new** IndexedModel();

IndexedModel normalModel = **new** IndexedModel();

HashMap<OBJIndex, Integer> resultIndexMap = **new** HashMap<OBJIndex, Integer>();

HashMap<Integer, Integer> normalIndexMap = **new** HashMap<Integer, Integer>();

HashMap<Integer, Integer> indexMap = **new** HashMap<Integer, Integer>();

**for** (**int** i = 0; i < **this**.\_indices.size(); i++)

{

OBJIndex currentIndex = **this**.\_indices.get(i);

Vector3f currentPosition = **this**.\_positions.get(currentIndex.\_vertexIndex);

Vector2f currentTexture;

Vector3f currentNormal;

**if**(**this**.\_hasTextureCoordinates)

currentTexture = **this**.\_textCoordinates.get(currentIndex.\_textureCoordIndex);

**else** currentTexture = Vector2f.*ZERO*;

**if**(**this**.\_hasNormals)

currentNormal = **this**.\_normals.get(currentIndex.\_normalIndex);

**else** currentNormal = Vector3f.*ZERO*;

Integer modelVertexIndex = resultIndexMap.get(currentIndex);

**if**(modelVertexIndex == **null**)

{

modelVertexIndex = res.getPositions().size();

resultIndexMap.put(currentIndex, modelVertexIndex);

res.getPositions().add(currentPosition);

res.getTextureCoordinates().add(currentTexture);

**if**(\_hasNormals)

res.getNormals().add(currentNormal);

}

Integer normalModelIndex = normalIndexMap.get(currentIndex.\_vertexIndex);

**if**(normalModelIndex == **null**)

{

normalIndexMap.put(currentIndex.\_vertexIndex, normalModel.getPositions().size());

normalModelIndex = normalModel.getPositions().size();

normalModel.getPositions().add(currentPosition);

normalModel.getTextureCoordinates().add(currentTexture);

normalModel.getNormals().add(currentNormal);

}

res.getIndices().add(modelVertexIndex);

normalModel.getIndices().add(normalModelIndex);

indexMap.put(modelVertexIndex, normalModelIndex);

}

**if**(!\_hasNormals)

{

normalModel.calcNormals();

**for** (**int** i = 0; i < res.getPositions().size(); i++)

res.getNormals().add(normalModel.getNormals().get(indexMap.get(i)));

}

**return** res;

}

**private** OBJIndex parseOBJIndex(String token)

{

String[] values = token.split("/");

OBJIndex res = **new** OBJIndex();

res.\_vertexIndex = Integer.*parseInt*(values[0]) - 1;

**if**(values.length > 1)

{

**this**.\_hasTextureCoordinates = **true**;

res.\_textureCoordIndex = Integer.*parseInt*(values[1]) - 1;

**if**(values.length > 2)

{

**this**.\_hasNormals = **true**;

res.\_normalIndex = Integer.*parseInt*(values[2]) - 1;

}

}

**return** res;

}

**public** ArrayList<Vector3f> getPositions() { **return** \_positions; }

**public** ArrayList<Vector2f> getTextureCoordinates() { **return** \_textCoordinates; }

**public** ArrayList<Vector3f> getNormals() { **return** \_normals; }

**public** ArrayList<OBJIndex> getIndices() { **return** \_indices; }

}

מחלקת התאורה הבסיסית

**public** **class** BaseLight **extends** GameComponent

{

**private** Vector3f \_color; //צבע התאורה

**private** **float** \_intensity; // חוזק התאורה

**private** Shader \_shader; //סוג התאורה

**public** BaseLight(Vector3f color, **float** intensity)

{

**this**.\_color = color;

**this**.\_intensity = intensity;

}

@Override

**public** **void** addToRenderingEngine(RenderingEngine engine)

{

engine.addLight(**this**);

}

**protected** **void** setShader(Shader shader) { **this**.\_shader = shader; }

**public** Shader getShader() { **return** **this**.\_shader; }

**public** Vector3f getColor() { **return** \_color; }

**public** **void** setColor(Vector3f \_color) { **this**.\_color = \_color; }

**public** **float** getIntensity() { **return** \_intensity; }

**public** **void** setIntensity(**float** \_intensity) { **this**.\_intensity = \_intensity;}

@Override

**public** **void** input(**float** delta) {

}

@Override

**public** **void** update(**float** delta) {

}

@Override

**public** **void** render(Shader shader, RenderingEngine engine){

}

}

מחלקת המצלמה

**public** **class** Camera **extends** GameComponent

{

**public** **static** **final** Vector3f ***yAxis*** = **new** Vector3f(0,1,0); //אובייקט שפשוט מגדיר את ציר ה- Y לחישובים מסוימים

**private** **boolean** \_mouse = **false**; //מגדיר שימוש בעכבר

**public** Matrix4f \_projection; //האזור שעליו המודלים יצויירו (נקודת המבט של המצלמה)

**public** Vector3f movementVector;

**private** Vector3f collisionVector;

**private** Vector3f oldPos;

**private** Vector3f newPos;

**public** Camera(**float** fov, **float** aspectRatio, **float** zNear, **float** zFar)

{

**this**.\_projection = **new** Matrix4f().initPerspective(fov, aspectRatio, zFar, zNear);

**this**.movementVector = **new** Vector3f();

**this**.setCollisionVector(**new** Vector3f());

**this**.setNewPos(**new** Vector3f());

**this**.setOldPos(**new** Vector3f());

}

**public** Matrix4f getViewProjection()

{

Matrix4f cameraRotation = getTransform().getTransformedRotation().conjugate().toRotationMatrix();

Vector3f cameraPosition = getTransform().getTransformedPosition().mul(-1);

Matrix4f cameraTranslation = **new** Matrix4f().initTranslation(cameraPosition.getX(),

cameraPosition.getY(),

cameraPosition.getZ());

**return** \_projection.mul(cameraRotation.mul(cameraTranslation));

}

Vector2f centerPosition = **new** Vector2f(Window.*getWidth*() / 2, Window.*getHeight*() / 2);

@Override

**public** **void** input(**float** delta)

{

**this**.movementVector = **new** Vector3f();

**if**(Input.*getKey*(Keyboard.***KEY\_D***))

**this**.movementVector = **this**.movementVector.add(**this**.getTransform().getRotation().getRight());

**if**(Input.*getKey*(Keyboard.***KEY\_A***))

**this**.movementVector = **this**.movementVector.add(**this**.getTransform().getRotation().getLeft());

**if**(Input.*getKey*(Keyboard.***KEY\_W***))

**this**.movementVector = **this**.movementVector.add(**this**.getTransform().getRotation().getForward());

**if**(Input.*getKey*(Keyboard.***KEY\_S***))

**this**.movementVector = **this**.movementVector.sub(**this**.getTransform().getRotation().getForward());

**if**(Input.*getMouseDown*(1))

{

\_mouse = !\_mouse;

Input.*setMousePos*(centerPosition);

Input.*setCursor*(!\_mouse);

}

**if**(\_mouse)

{

Vector2f deltaPos = Input.*getMousePos*().sub(centerPosition);

**boolean** rotY = deltaPos.getX() != 0;

**boolean** rotX = deltaPos.getY() != 0;

**if**(rotY)

**this**.getTransform().rotate(***yAxis***, deltaPos.getX() \* 0.005f);

**if**(rotX)

**this**.getTransform().rotate(**this**.getTransform().getRotation().getRight(), -deltaPos.getY() \* 0.005f);

**if**(rotX || rotY)

Input.*setMousePos*(centerPosition);

}

}

@Override

**public** **void** update(**float** delta)

{

**float** movAmt;

**if**(Input.*getKey*(Keyboard.***KEY\_LSHIFT***) || Input.*getKey*(Keyboard.***KEY\_RSHIFT***))

movAmt =(**float**)(40 \* delta) \* 1.5f;

**else** movAmt = (**float**)(40 \* delta);

**this**.movementVector.setY(0);

oldPos = **this**.getTransform().getPosition();

newPos = oldPos.add(movementVector.mul(movAmt));

movementVector = movementVector.mul(collisionVector);

**if**(movementVector.length() > 0)

**this**.move(movementVector, movAmt);

}

**public** **void** move(Vector3f direction, **float** amount)

{

**this**.getTransform().setPosition(**this**.getTransform().getPosition().add(direction.mul(amount)));

}

@Override

**public** **void** addToRenderingEngine(RenderingEngine engine)

{

engine.addCamera(**this**);

}

@Override

**public** **void** render(Shader shader, RenderingEngine engine) {

}

**public** Vector3f getCollisionVector() {

**return** collisionVector;

}

**public** **void** setCollisionVector(Vector3f collisionVector) {

**this**.collisionVector = collisionVector;

}

**public** Vector3f getNewPos() {

**return** newPos;

}

**public** **void** setNewPos(Vector3f newPos) {

**this**.newPos = newPos;

}

**public** Vector3f getOldPos() {

**return** oldPos;

}

**public** **void** setOldPos(Vector3f oldPos) {

**this**.oldPos = oldPos;

}

**public** Vector3f getMovementVector() {

**return** movementVector;

}

**public** **void** setMovementVector(Vector3f movementVector) {

**this**.movementVector = movementVector;

}

}

מחלקת העזר לתאורה ישירה

**public** **class** DirectionalLight **extends** BaseLight

{

**public** DirectionalLight(Vector3f color, **float** intensity)

{

**super**(color, intensity);

setShader(ForwardDirectional.*getInstance*());

}

\

**public** Vector3f getDirection() { **return** getTransform().getTransformedRotation().getForward(); }

}

מחלקת ציור המודל

**public** **class** MeshRenderer **extends** GameComponent

{

**private** Mesh \_mesh; // מודל

**private** Material \_material; //חומר המודל

**public** MeshRenderer(Mesh mesh, Material material)

{

**this**.\_mesh = mesh;

**this**.\_material = material;

}

**public** **void** render(Shader shader, RenderingEngine engine)

{

shader.bind();

shader.updateUniforms(getTransform(), \_material, engine);

\_mesh.render();

}

@Override

**public** **void** input(**float** delta) {}

@Override

**public** **void** update(**float** delta) {}

@Override

**public** **void** addToRenderingEngine(RenderingEngine engine) {}

}

מחלקת עזר לתאורה הנקודתית

**public** **class** PointLight **extends** BaseLight

{

**private** **static** **final** **int** ***COLOR\_DEPTH*** = 256; //תמיכה רק ב256 צבעים שנצטרך לתאורה

**private** **float** \_range; //מקסימום טווח

**private** Vector3f \_attenuation; // ההדללה של התאורה בכל ציר

**public** PointLight(Vector3f color, **float** intensity, Vector3f attenuation)

{

**super**(color, intensity);

**this**.\_attenuation = attenuation;

**float** a = **this**.\_attenuation.getZ();

**float** b = **this**.\_attenuation.getY();

**float** c = **this**.\_attenuation.getX() - PointLight.***COLOR\_DEPTH*** \* getIntensity() \* getColor().max();

**this**.setRange((**float**)(-b + Math.*sqrt*(b \* b - 4 \* a - c)) / (2 \* a));

**this**.setShader(ForwardPoint.*getInstance*());

}

**public** **float** getRange() { **return** \_range; }

**public** **void** setRange(**float** range) { **this**.\_range = range; }

**public** **float** getConstant() { **return** **this**.\_attenuation.getX(); }

**public** **float** getLinear() { **return** **this**.\_attenuation.getY(); }

**public** **float** getExponent() { **return** **this**.\_attenuation.getZ(); }

**public** **void** setConstant(**float** constant) { **this**.\_attenuation.setX(constant); }

**public** **void** setLinear(**float** linear) { **this**.\_attenuation.setY(linear); }

**public** **void** setExponent(**float** exponent) { **this**.\_attenuation.setZ(exponent); }

}

מחלקת התאורה הנקודתית המשודרגת

**public** **class** SpotLight **extends** PointLight

{

**private** **float** \_cutoff; //הורדת כמות התאורה לכל פיקסל

**public** SpotLight(Vector3f color, **float** intensity, Vector3f attenuation, **float** cutoff)

{

**super**(color, intensity, attenuation);

**this**.\_cutoff = cutoff;

**this**.setShader(ForwardSpot.*getInstance*());

}

**public** Vector3f getDirection() {

**return** getTransform().getTransformedRotation().getForward();

}

**public** **float** getCutoff() { **return** \_cutoff; }

**public** **void** setCutoff(**float** cutoff) { **this**.\_cutoff = cutoff; }

}

מחלקת המאפיין של אובייקט המשחק

**public** **abstract** **class** GameComponent

{

**private** GameObject \_parent; //האובייקט ששולט במחלקה

**public** **abstract** **void** input(**float** delta);

**public** **abstract** **void** update(**float** delta);

**public** **abstract** **void** render(Shader shader, RenderingEngine engine);

**public** **abstract** **void** addToRenderingEngine(RenderingEngine engine);

**public** **void** setParent(GameObject parent) { **this**.\_parent = parent; }

**public** Transform getTransform() { **return** **this**.\_parent.getTransform(); }

}

מחלקת התפריט

**enum** EntryState

{

***Begin***,

***Operational***,

***Finishing***,

***Done***

}

**public** **abstract** **class** Menu

{

**private** List<Entry> \_entries;

**protected** **int** \_currentEntry;

**protected** EntryState \_state = EntryState.***Begin***;

**private** **float** \_fade = 0;

**private** **boolean** \_allowInput;

**public** Menu()

{

init();

}

**protected** **void** init()

{

**this**.\_entries = **new** Vector<Entry>();

Entry.*setFont*(Menu.*loadFont*("Calibri", Font.***ITALIC***, 24));

**this**.\_currentEntry = 0;

**this**.\_state = EntryState.***Begin***;

addEntries();

**this**.\_allowInput = **true**;

}

**protected** **void** disableInput()

{

**this**.\_allowInput = **false**;

}

**protected** **void** addEntry(String label)

{

**if**(**this**.\_entries.size() == 0)

Entry.*setID*(0);

**this**.\_entries.add(**new** Entry(label));

}

**protected** **abstract** **void** addEntries();

**void** getInput()

{

**while**(Keyboard.*next*())

{

**if**(Keyboard.*getEventKeyState*())

{

**if**(Keyboard.*getEventKey*() == Keyboard.***KEY\_UP***)

{

**this**.\_currentEntry--;

**if**(0>**this**.\_currentEntry) **this**.\_currentEntry = **this**.\_entries.size()-1;

}

**if**(Keyboard.*getEventKey*() == Keyboard.***KEY\_DOWN***)

{

**this**.\_currentEntry++;

**if**(**this**.\_currentEntry > **this**.\_entries.size() - 1) **this**.\_currentEntry = 0;

}

**if**(Keyboard.*getEventKey*() == Keyboard.***KEY\_RETURN***)

{

**this**.\_state = EntryState.***Finishing***;

}

}

}

**int** x = Mouse.*getX*();

**int** y = Window.*getHeight*() - Mouse.*getY*();

Rectangle mouseRec = **new** Rectangle(x, y, 1, 1);

**for**(Entry e : **this**.\_entries)

{

**if**(e.getRectangle().contains(mouseRec))

**this**.\_currentEntry = e.getId();

//System.out.println(mouseRec);

}

**if**(Input.*getMouseDown*(1))

{

**this**.\_state = EntryState.***Finishing***;

}

}

**public** **void** update()

{

**if**(**this**.\_entries.size() != 0)

{

**for**(**int** i = 0; i<**this**.\_entries.size(); i++)

{

**if**(**this**.\_entries.get(i).id == **this**.\_currentEntry)

{

**this**.\_entries.get(i).setChosen(**true**);

**this**.\_entries.get(i).setColor(**new** Color(.65f, .124f, .132f, 1f));

}

**else**

{

**this**.\_entries.get(i).setChosen(**false**);

**this**.\_entries.get(i).setColor(Color.***gray***);

}

}

**switch**(**this**.\_state)

{

**case** ***Begin***:

**if**(90 > **this**.\_fade) **this**.\_fade += 1f;

**else** **this**.\_state = EntryState.***Operational***;

**break**;

**case** ***Operational***:

**if**(**this**.\_allowInput) **this**.getInput();

**break**;

**case** ***Finishing***:

**if**(**this**.\_fade > 0) **this**.\_fade -= 1f;

**else** **this**.\_state = EntryState.***Done***;

**break**;

**case** ***Done***:**this**.init();

**break**;

}

}

}

**public** **void** render()

{

**for**(Entry e : **this**.\_entries)

{

e.setColor(**new** Color(e.getColor().r, e.getColor().g, e.getColor().b, (**float**) Math.*sin*(Math.*toRadians*(\_fade))));

Statics.*renderText*(Entry.*getFont*(), e.getName(), e.getPosition(), e.getColor());

}

}

**public** **static** TrueTypeFont loadFont(String name, **int** attribute, **int** size)

{

Font awtFont = **new** Font(name, attribute, size);

**return** **new** TrueTypeFont(awtFont, **true**);

}

}

מחלקת התוית

**public** **class** Entry

{

**private** **static** **int** *ID*=0;

**public** **int** id = 0;

**private** **static** Vector2f *startPos* = **new** Vector2f(Display.*getWidth*() / 2, Display.*getHeight*() / 2);

**private** **static** TrueTypeFont *font*;

**private** Vector2f position;

**private** String name;

**private** **boolean** chosen;

**private** org.newdawn.slick.Color color;

**private** Rectangle rectangle;

**public** **static** **float** *fade*;

**public** Entry(String name)

{

**this**.name = name;

**this**.chosen = **false**;

**this**.color = org.newdawn.slick.Color.***gray***;

*fade* = 0;

**this**.id = *ID*;

*ID*++;

**this**.position = **new** Vector2f(*startPos*.x, *startPos*.y + (**this**.id \* 40));

**this**.rectangle = **new** Rectangle((**int**)**this**.position.x, (**int**)**this**.position.y, *font*.getWidth(**this**.name), *font*.getHeight(**this**.name));

}

**public** **static** **int** getID() {

**return** *ID*;

}

**public** **static** **void** setID(**int** iD) {

*ID* = iD;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** **static** Vector2f getStartPos() {

**return** *startPos*;

}

**public** **static** **void** setStartPos(Vector2f startPos) {

Entry.*startPos* = startPos;

}

**public** **static** TrueTypeFont getFont() {

**return** *font*;

}

**public** **static** **void** setFont(TrueTypeFont font) {

Entry.*font* = font;

}

**public** Vector2f getPosition() {

**return** position;

}

**public** **void** setPosition(Vector2f position) {

**this**.position = position;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **boolean** isChosen() {

**return** chosen;

}

**public** **void** setChosen(**boolean** chosen) {

**this**.chosen = chosen;

}

**public** org.newdawn.slick.Color getColor() {

**return** color;

}

**public** **void** setColor(org.newdawn.slick.Color white) {

**this**.color = white;

}

**public** Rectangle getRectangle() {

**return** rectangle;

}

**public** **void** setRectangle(Rectangle rectangle) {

**this**.rectangle = rectangle;

}

}

מחלקת התפריט הראשי

**public** **class** MainMenu **extends** Menu

{

**public** MainMenu()

{

**super**();

}

**protected** **void** addEntries()

{

**super**.addEntry("Play Story");

**super**.addEntry("Instructions");

**super**.addEntry("Options");

**super**.addEntry("Quit");

}

**public** **void** update()

{

**super**.update();

**if**(**super**.\_state == EntryState.***Done***)

{

**switch**(**super**.\_currentEntry)

{

**case** 0: PlaySelected(); **break**;

**case** 1: InstructionsSelected(); **break**;

**case** 2: OptionsSelected(); **break**;

**case** 3: QuitSelected(); **break**;

}

//this.init();

}

}

**private** **void** PlaySelected()

{

MenuManager.*menuState* = MenuState.***StartGame***;

MenuManager.*gameState* = GameState.***Loading***;

}

**private** **void** InstructionsSelected()

{

MenuManager.*menuState* = MenuState.***Instructions***;

}

**private** **void** OptionsSelected()

{

MenuManager.*menuState* = MenuState.***Options***;

}

**private** **void** QuitSelected()

{

CoreEngine.*cleanUp*();

}

}

מחלקת ניהול התפריטים

**public** **class** MenuManager

{

**public** **static** GameState *gameState*;

**public** **static** MenuState *menuState*;

**private** **static** MainMenu *mainMenu*;

**private** **static** Options *options*;

**private** **static** Instructions *instructions*;

**private** **static** Loading *loading*;

**private** **static** Pause *pause*;

**public** **static** **void** init()

{

*gameState* = GameState.***MainMenu***;

*menuState* = MenuState.***MainMenu***;

*mainMenu* = **new** MainMenu();

*options* = **new** Options();

*instructions* = **new** Instructions();

*loading* = **new** Loading();

*pause* = **new** Pause();

}

**public** **static** **void** update()

{

RenderingEngine.*clrScreen*();

**if** (MenuManager.*gameState* == GameState.***MainMenu*** && MenuManager.*menuState* == MenuState.***MainMenu***)

*mainMenu*.update();

**if** (MenuManager.*menuState* == MenuState.***Options*** && MenuManager.*gameState* == GameState.***MainMenu***)

*options*.update();

**if**(MenuManager.*menuState* == MenuState.***Instructions*** && *gameState* == GameState.***MainMenu***)

*instructions*.update();

**if**(MenuManager.*menuState* == MenuState.***StartGame*** && *gameState* == GameState.***Loading***)

*loading*.update();

**if**(MenuManager.*gameState* == GameState.***Paused*** && MenuManager.*menuState* == MenuState.***StartGame***)

*pause*.update();

}

**public** **static** **void** render()

{

**if** (MenuManager.*gameState* == GameState.***MainMenu*** && MenuManager.*menuState* == MenuState.***MainMenu***)

*mainMenu*.render();

**if** (MenuManager.*menuState* == MenuState.***Options*** && MenuManager.*gameState* == GameState.***MainMenu***)

*options*.render();

**if**(MenuManager.*menuState* == MenuState.***Instructions*** && *gameState* == GameState.***MainMenu***)

*instructions*.render();

**if**(MenuManager.*menuState* == MenuState.***StartGame*** && *gameState* == GameState.***Loading***)

*loading*.render();

**if**(MenuManager.*gameState* == GameState.***Paused*** && MenuManager.*menuState* == MenuState.***StartGame***)

*pause*.render();

}

}

מחלקת ארגון האובייקטים במשחק

**public** **class** Arrange

{

**public** **static** ArrayList<GameObject> arrangeTrees(GameObject parent, **int** terrainWidth, **int** terrainDepth)

{

GameObject tree[][] = **new** GameObject[10][5];

ArrayList<GameObject>trees = **new** ArrayList<GameObject>();

Mesh treeModel = **new** Mesh("low poly tree.obj");

Material treeMaterial = **new** Material();

treeMaterial.addTexture("diffuse", Texture.*loadTextureID*("tree.jpg"));

treeMaterial.addFloat("specularIntensity", 0.5f);

treeMaterial.addFloat("specularPower", 6f);

Random rnd = **new** Random();

**for** (**int** i = 0; i < tree.length; i++) {

**for** (**int** j = 0; j < tree[i].length; j++)

{

tree[i][j] = **new** GameObject();

tree[i][j].addComponent(**new** MeshRenderer(treeModel, treeMaterial));

tree[i][j].getTransform().setScale(**new** Vector3f(1, 1, 1));

tree[i][j].getTransform().setPosition(**new** Vector3f(rnd.nextInt(terrainWidth \* 3) - terrainWidth, tree[i][j].getTransform().getScale().getY(), rnd.nextInt(terrainDepth \* 3) - terrainDepth));

**if**(tree[i][j].getTransform().getPosition().getX() >= -terrainWidth / 3 &&

tree[i][j].getTransform().getPosition().getX() <= terrainWidth / 3 &&

tree[i][j].getTransform().getPosition().getZ() >= -terrainDepth / 3 &&

tree[i][j].getTransform().getPosition().getZ() <= terrainDepth / 3)

{

System.***out***.println(tree[i][j].getTransform().getPosition().getXZ());

trees.add(tree[i][j]);

parent.addChild(tree[i][j]);

}

**else** j--;

}

}

**return** trees;

}

}

מחלקת התכלס משחק

**public** **class** FinalGame **extends** Game

{

PhysicsEngine physicsEngine;

**private** GameObject cameraObject;

**private** Camera camera;

**private** PhysicsObject camShpere;

**private** SpotLight flashlightComponent;

**private** GameObject flashLight;

**private** **boolean** isFlashOn;

**private** **float** flashIntensity;

**private** GameObject terrain;

**private** Material forestGrass;

**private** **float** terrainDepth = 1000.0f;

**private** **float** terrainWidth = 1000.0f;

**private** ArrayList<GameObject> trees;

**private** **int** difficulty;

**float** temp = 0;

**public** FinalGame()

{

**this**.physicsEngine = **new** PhysicsEngine();

**this**.camera = **new** Camera((**float**)Math.*toRadians*(40f), Window.*getWidth*() / Window.*getHeight*(), 0.01f, 1000f);

**this**.cameraObject = **new** GameObject().addComponent(camera);

**this**.isFlashOn = **true**;

**this**.flashLight = **new** GameObject();

**this**.terrain = **new** GameObject();

**this**.forestGrass = **new** Material();

**this**.trees = **new** ArrayList<GameObject>();

}

**public** **void** init()

{

MenuManager.*init*();

**this**.difficulty = 1;

**this**.cameraObject.getTransform().setPosition(**new** Vector3f(0, 4, 0));

**this**.flashlightComponent = **new** SpotLight(**new** Vector3f(0.2f, 0.2f, 0.2f), 1.4f,

**new** Vector3f(0, 0, 0.1f), //Attenuation

0.9f);

**this**.flashIntensity = **this**.flashlightComponent.getIntensity();

**this**.flashLight.addComponent(flashlightComponent);

**this**.flashLight.getTransform().getPosition().setZ(**this**.flashLight.getTransform().getPosition().getZ() + 14);

**this**.forestGrass.addTexture("diffuse", Texture.*loadTextureID*("forestgrass1.jpg"));

**this**.forestGrass.addFloat("specularIntensity", 0.5f);

**this**.forestGrass.addFloat("specularPower", 6f);

**this**.terrainDepth = 1000.0f;

**this**.terrainWidth = 1000.0f;

Vertex[] terrainVertices = **new** Vertex[] {**new** Vertex(**new** Vector3f(-terrainWidth, 0.0f, -terrainDepth), **new** Vector2f(0,0)),

**new** Vertex(**new** Vector3f(-terrainWidth, 0.0f, terrainDepth \* 3), **new** Vector2f(0.0f, 1.0f)),

**new** Vertex(**new** Vector3f(terrainWidth \* 3, 0.0f, -terrainDepth), **new** Vector2f(1.0f, 0)),

**new** Vertex(**new** Vector3f(terrainWidth \* 3, 0.0f, terrainDepth \* 3), **new** Vector2f(1.0f,1.0f))};

**int**[] terrainIndices = **new** **int**[] {0, 1, 2,

2, 1, 3};

**this**.terrain.addComponent(**new** MeshRenderer(**new** Mesh(terrainVertices, terrainIndices), forestGrass));

**this**.terrain.setImportant(**true**);

**this**.addToRoot(**this**.terrain);

**this**.trees = Arrange.*arrangeTrees*(**this**.terrain, (**int**)terrainWidth, (**int**)terrainDepth);

System.***out***.println(**this**.trees.size());

Page.*init*(**this**.trees);

Slender.*init*();

**this**.terrain.addChild(Slender.*getObject*());

**this**.terrain.addChild(cameraObject);

**this**.cameraObject.addChild(flashLight);

**this**.camShpere = **new** PhysicsObject(**new** BoundingSphere(**this**.cameraObject.getTransform().getPosition(), 0.2f), **this**.camera.getMovementVector());

**this**.physicsEngine.addObject(camShpere);

**for** (**int** i = 0; i < trees.size(); i++)

{

**this**.physicsEngine.addObject(**new** PhysicsObject(**new** BoundingSphere(trees.get(i).getTransform().getPosition(), 5f), Vector3f.*ZERO*));

}

PhysicsEngineComponent physicsEngineComponent = **new** PhysicsEngineComponent(**this**.physicsEngine);

**for** (**int** i = 0; i < physicsEngineComponent.getEngine().getNumObjects(); i++) {

**this**.addToRoot(**new** GameObject().addComponent(**new** PhysicsObjectComponent(physicsEngineComponent.getEngine().getObject(i))));

}

**this**.addToRoot(**new** GameObject().addComponent(physicsEngineComponent));

}

**public** **void** input(**float** delta)

{

**super**.input(delta);

**if**(Input.*getKeyDown*(Keyboard.***KEY\_F***))

{

**this**.isFlashOn = !**this**.isFlashOn;

**if**(**this**.isFlashOn) **this**.flashlightComponent.setIntensity(flashIntensity);

**else** **this**.flashlightComponent.setIntensity(-4f);

}

}

**public** **void** update(**float** delta)

{

**super**.update(delta);

cameraObject.getTransform().getPosition().setY(8);

camShpere.setVelocity(camera.getMovementVector());

camShpere.setPosition(cameraObject.getTransform().getPosition());

Slender.*update*(**this**.difficulty, cameraObject.getTransform().getPosition(), cameraObject.getTransform().getRotation());

**if**(**this**.isFlashOn)

{

**if**(**this**.flashlightComponent.getIntensity() > -1)

{

**this**.flashlightComponent.setIntensity(**this**.flashlightComponent.getIntensity() - delta/100);

**this**.flashIntensity = **this**.flashlightComponent.getIntensity();

}

**else** **this**.flashlightComponent.setIntensity(-4);

}

Vector3f collisionVector = **new** Vector3f(1f, 1f, 1f);

camera.setCollisionVector(collisionVector);

//if(Page.getLength() == 0) //GOTO: winState

doNothing();

**if**(**true**)

//Waited too much || Slender is near you for too long (depending on difficulty)

//GOTO: StaticScreen and from there loseState

doNothing();

//Borders

**this**.cameraObject.getTransform().getPosition().setX(Statics.*clamp*(

**this**.cameraObject.getTransform().getPosition().getX(),

-**this**.terrainWidth / 2, **this**.terrainWidth / 2));

**this**.cameraObject.getTransform().getPosition().setZ(Statics.*clamp*(

**this**.cameraObject.getTransform().getPosition().getZ(),

-**this**.terrainDepth / 2, **this**.terrainDepth / 2));

}

}

מחלקת הדפים

**public** **class** Page

{

**private** **static** List<GameObject> *\_pages*;

**private** **static** Material *\_material*;

**private** **static** Mesh *\_model*;

**public** **static** **void** init(ArrayList<GameObject> trees)

{

*\_pages* = **new** ArrayList<GameObject>();

*\_material* = **new** Material();

*\_material*.addFloat("specularIntensity", 0.5f);

*\_material*.addFloat("specularPower", 6f);

*\_material*.addTexture("diffuse", Texture.*loadTextureID*("forestgrass1.jpg"));

*\_model* = **new** Mesh("box.obj");

*initLocations*(trees);

}

**private** **static** **void** initLocations(ArrayList<GameObject> trees)

{

Random rnd = **new** Random();

**for** (**int** i = 0; i < 4; i++)

{

**int** index = rnd.nextInt(trees.size());

GameObject tree = trees.get(index);

*\_pages*.add(**new** GameObject());

*\_pages*.get(i).addComponent(**new** MeshRenderer(*\_model*, *\_material*));

*\_pages*.get(i).getTransform().getPosition().setY(8);

*\_pages*.get(i).getTransform().getPosition().setX(tree.getTransform().getTransformedPosition().getCopy().getX());

*\_pages*.get(i).getTransform().getPosition().setZ(tree.getTransform().getTransformedPosition().getCopy().getZ());

*\_pages*.get(i).getTransform().setScale(*\_pages*.get(i).getTransform().getScale().mul(4.75f));

trees.get(index).addChild(*\_pages*.get(i));

System.***out***.println("page " + i + " is added at " + *\_pages*.get(i).getTransform().getPosition());

}

}

**public** **static** **boolean** removePage(GameObject m)

{

**return** *\_pages*.remove(m);

}

**public** **static** **int** getLength() { **return** *\_pages*.size(); }

**public** **static** GameObject getPage(**int** index) { **return** *\_pages*.get(index); }

**public** **static** List<GameObject> getPages() { **return** *\_pages*; }

}

מחלקת הסלנדר

**public** **class** Slender

{

**private** **static** GameObject *object*; //האובייקט שעליו יישבו כל האפשרויות

**private** **static** Material *material*; //סוג החומר והטקסטורה של הסלנדר

**private** **static** **int** *timer*;//כל כמה שניות סלנדר ישתגר למיקום של השחקן

**private** **static** **boolean** *isVisible*;//סלנדר ישתגר רק כשלא רואים אותו

**private** **static** **boolean** *isInRnage*;//אם הסלנדר לא יותר מידי קרוב הוא ישתגר

**private** **static** Random *rnd*; //(ראנדום)

**public** **static** **void** init()

{

*setObject*(**new** GameObject());

*setMaterial*(**new** Material());

*material*.addTexture("diffuse", Texture.*loadTextureID*("slender.PNG"));

*material*.addFloat("specularIntensity", 0.5f);

*material*.addFloat("specularPower", 6f);

*object*.addComponent(**new** MeshRenderer(**new** Mesh("Slenderman Model.obj"), *material*));

*object*.getTransform().setPosition(**new** Vector3f(250, 10, 150));

*object*.getTransform().setRotation(**new** Quaternion(**new** Vector3f(0, 1, 0), (**float**)Math.*toRadians*(-90)));

*object*.getTransform().setScale(**new** Vector3f(0.03f, 0.03f, 0.03f));

*timer* = 0;

*isVisible* = **false**;

*isInRnage* = **false**;

*rnd* = **new** Random();

}

**public** **static** **void** update(**float** difficulty, Vector3f playerPosition, Quaternion playerRotation)

{

**float** distance = Math.*abs*((-1) \* (**float**) Math.*pow*(Slender.*getObject*().getTransform().getPosition().getX() - playerPosition.getCopy().getX(), 2) +

(**float**) Math.*pow*(Slender.*getObject*().getTransform().getPosition().getZ() - playerPosition.getCopy().getZ(), 2));

*timer* = (**int**) Time.*getTime*() % 100;

**if**((*timer* % ((difficulty / 4 ) \* 60)) == 0)

{

*checkIfVisible*();

**if**(!*isVisible*())

{

**if**(distance > 7500)

{

System.***out***.println(distance);

Slender.*object*.getTransform().setPosition(playerPosition.getCopy().sub(**new** Vector3f(*rnd*.nextInt(300) - 150,0,*rnd*.nextInt(300) - 150)));

Slender.*object*.getTransform().getPosition().setY(10f);

Vector3f positionToCheck = **new** Vector3f(playerPosition.getX(), Slender.*object*.getTransform().getPosition().getY(), playerPosition.getZ());

Slender.*object*.getTransform().setRotation(Slender.*object*.getTransform().getRotation().lookAt(positionToCheck, **new** Vector3f(0,1,0)));

}

}

}

**if**(distance < 4000) *isInRnage* = **true**;

**else** *isInRnage* = **false**;

}

**private** **static** **void** checkIfVisible()

{

}

}