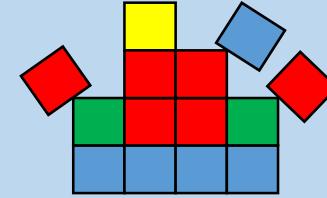


# Build & Destroy



## Helicopter simulator

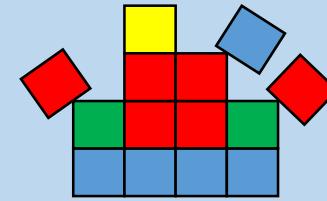
&

## Strategy

המכללה האקדמית תל-חי

מגיש : פואד חורי - 315954834

מנחה : תמר צמח



# Build & Destroy

Build & Destroy

## תוכן עניינים:

<b>Object &amp; Map Editor :</b>	<b>9</b>
<b>9.1 :</b>	<b>מבוא</b>
<b>9.2 :</b>	<b>ממשק משתמש ו</b>
<b>9.3 :</b>	<b>מחלקות</b>
<b>9.4 :</b>	<b>שמירת מפה</b>
<b>9.5 :</b>	<b>תרשים השתלטות השחקן</b>
<b>UML :</b>	<b>9.6</b>
<b>UI Management :</b>	<b>10</b>

<b>1 :</b>	<b>מבוא</b>
<b>1.1 :</b>	<b>חוקים</b>
<b>1.2 :</b>	<b>כליים</b>
<b>1.3 :</b>	<b>OpenGL</b>
<b>2 :</b>	<b>תרשים השתלטות השחקן</b>
<b>3 :</b>	<b>ו UI ממשק משתמש</b>
<b>4 :</b>	<b>מחלקות</b>
<b>5 :</b>	<b>אלגוריתמיקה</b>
<b>6 :</b>	<b>чисובים מתמטיים</b>
<b>7 :</b>	<b>התנגשות</b>
<b>8 :</b>	<b>UML</b>

# **מבוא**

## **חוקים & כלים**

### **המשחק:**

הוא שחזור אחד מול המחשב, בטור התחלת כל שחזור יש: (בנין שלא מבוקש או אפשרות לבנות תנאים ולפזר אותם במפה).  
הבניין הראשי של כל שחזור יהיה שונים בצדדים שונים במפה.

### **מטרת המשחק היא "משימה ראשית":** לבנות הבניין שלו,

השחקן אמור להביא בלוקים מהמפה שבעזרתם יבנה את הבניין שלו.  
השחקן מהתשלט על המוסוק ובעזרתו הוא תופס בלוק מהמפה וחוזר בבלוק אל הבניין שלו וזרק את הבלוק כר שילבש על הבניין.

הבלוקים שבעזרתם השחקן בונה את הבניין נמצאים בשלושה תחנותות שמייצרות בלוקים.  
השחקן בזמן איסוף הבלוקים במסוק אפשר גם לאסוף כר שייהי יכול לקנות תנאים.

### **משימות משנהיות:**

1. לשמר על הבניין שלו בעזרת תנאים.
2. להכשיל את השחקן השני.
3. להרוויח הבניין של השחקן השני בעזרת תנאים.

### **התשלטות השחקן היא :**

1. נהיגת את המוסוק בעזרת העכבר והמקלדת.
2. לפזר תנאים במפה ולתת להם הוראות בעזרת העכבר והמקלדת.

# **מבוא**

## **חוקים & כלים**

### **הוראות לטנקים:**

1. לשמר על הבניין הראשי.
2. לעקוב אחרי המסוק של השחקן השני.
3. להגן על המסוק.
4. להגן על טנק אחר.
5. להרוויח הבניין הראשי של השחקן השני.
6. לשלוח את הטנק לאיזה שהוא מקום במטה ולשמור עליו.

### **לכל טנק יש משימה מסוימת:**

טנק 1 : משימתו לתקוף את הטנקים של השחקן השני. "ירוה פגזים"

טנק 2 : משימתו לתקוף מסוק וטילים של השחקן השני. "ירוה כדורים"

טנק 3 : משימתו לתקוף מסוק של השחקן השני. "ירוה שני טילים שעוקבים אחרי המסוק"

טנק 4 : משימתו לתקוף הבניין הראשי של השחקן השני. "ירוה טילים מרוחק"

**במקרה והשחקן איבד את המסוק:** הוא פשוט יחזור לבניין הראשי שלו.

**במקרה והשחקן איבד את הטנקים:** הוא יכול לקנות.

# מבוא

## חוקים & כלים

### המחשב:

יש אפשרות לבחור קושה של המחשב: (קל \ בינוני \ קשה).

שהמחשב ישחק ויש לו שלושה תוכניות לשחק בהן:

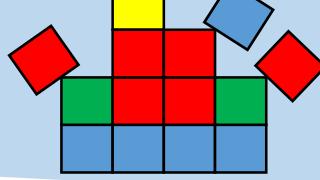
- 1. תוכנית התקפה :** תוכנית שבו המחשב ינסה להרווș לשחקן את הבניין שלו, בעזרת טנקים.  
מחשב יבחר לשחק בתוכנית זו בזמן שהשחקן מתקרב לסיים את הבניין שלו.

- 2. תוכנית רגילה :** תוכנית שבו המחשב מפזר את הטנקים שלו באמצעות המפה להרווș את המסוק והטנקים של השחקן.  
מחשב יבחר לשחק בתוכנית זו בזמן שאין הפרש גדול בין הבניינים של השחקן והמחשב ואף אחד לא מתקרב לסיים.

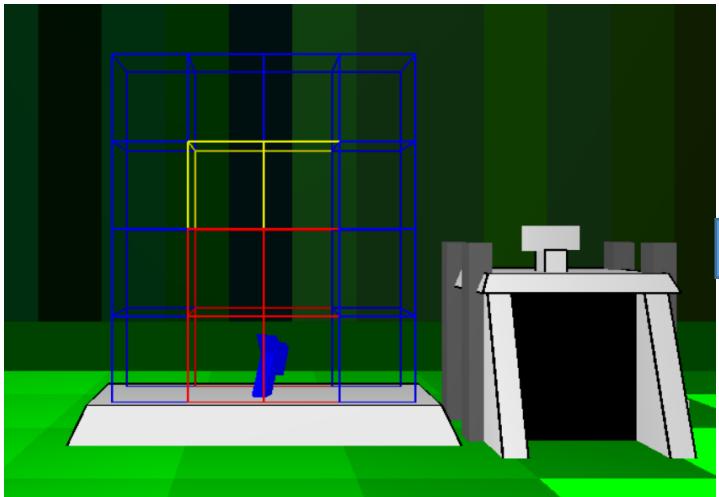
- 3. תוכנית הגנה :** תוכנית שבו המחשב מפזר כל הטנקים שלו להגן על הבניין שלו.  
מחשב יבחר לשחק בתוכנית זו בזמן שהמחשב מתקרב לסיים את הבניין שלו.

ובכל תוכנית שהמחשב ישחק ויפזר הטנקים בצורה אחרת לפי הקושה שהשחקן יבחר.

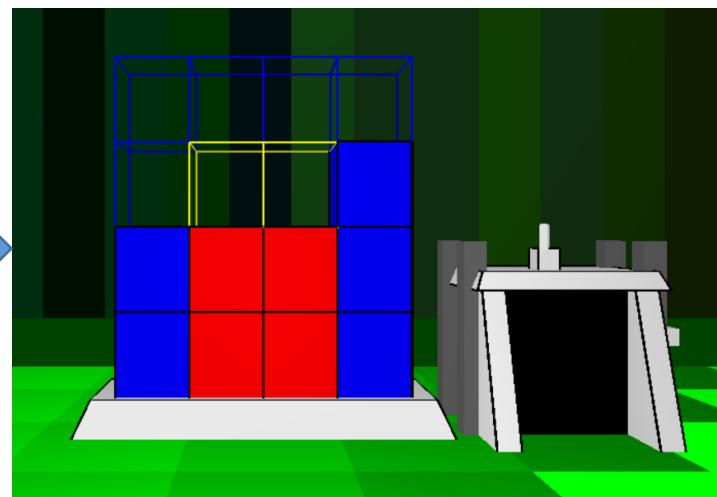
**מבוא  
חוקים & כלים**



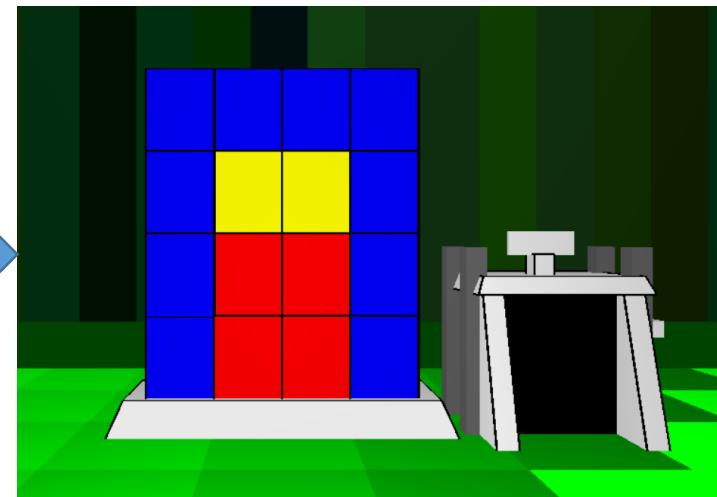
**מטרה:**



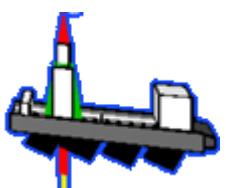
**0/16**



**9/16**



**16/16  
Wins**



**坦ク 4**



**坦ク 3**



**坦ク 2**



**坦ク 1**

**タンク :**

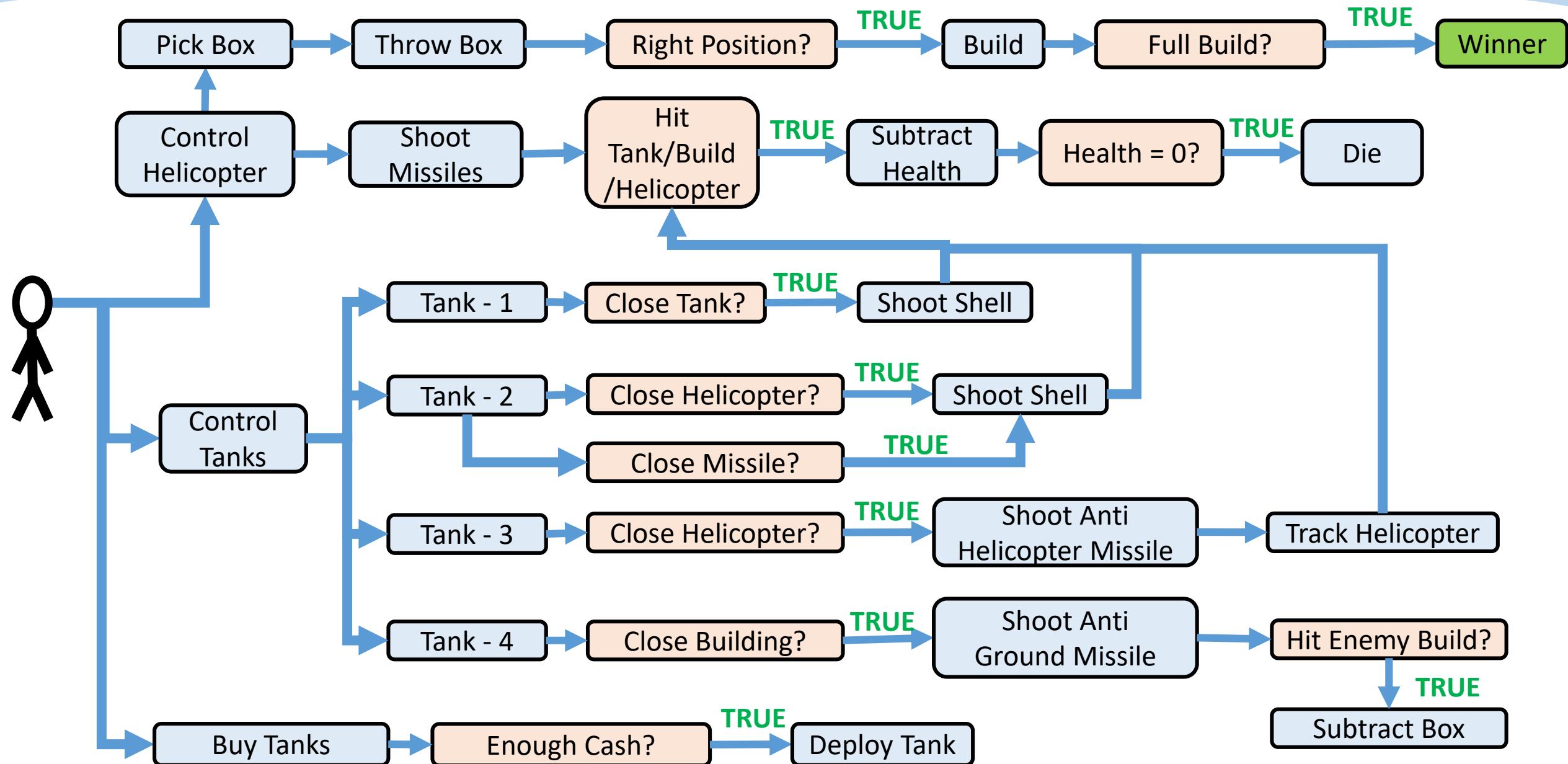
הוא ממשק תוכנית יישומים ליצירת תוכנות שמייצרות גרפיקה תלת מימדית ממוחשבת (וגם דו-ממדית). הממשק מורכב מיותר מ-250 קריאות פונקציות שונות שיכולות לשמש לצור של סצנות תלת-מימדיות מורכבות מצורות פשוטות (פרימיטיבים).

הפעולה הבסיסית של OpenGL היא לקבל סוגים שונים בסיסיים כמו נקודה, שורה, או פוליגון ולהמיר אותם לפיקסלים (רטראיזציה). המירה זאת נעשית בעזרת מכונות המצלבים של OpenGL רוב הפקודות של OpenGL עוסקות בהמרה זאת, או על ידי שליחת משתנים למוכנת המצלבים להמרה או בשליטה בקונפיגורציה של עיבוד הנתונים במוכנת המצלבים. לפני הופעת OpenGL 2.0 כל שלב של מכונת המצלבים ביצע פעולה אחת והוא לו אפשרות קונפיגורציה מוגבלות. ב-2.0 OpenGL יש יותר אפשרות לניטנות לעזרת .

**פונקציות שהשתמשתי :**

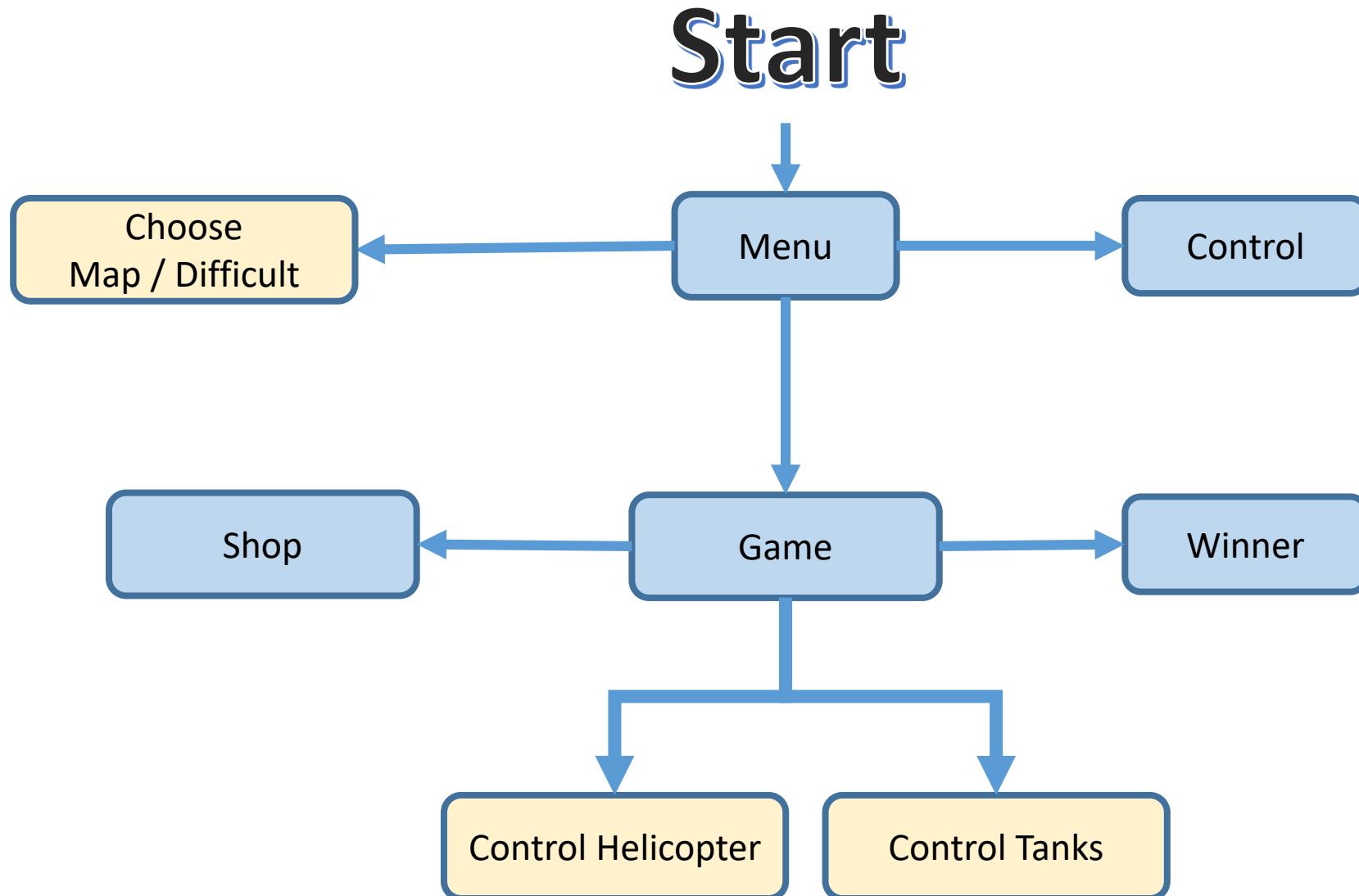
1. הzzת עבר.
2. לחיצת עבר.
3. לחיצר כפתור.
4. להזמת סימון סרטון הzzה יחסית בשלושה המימדים, "יש לשמר על נקודה ראשית כך שכל האובייקטים יהיו עם מיקום ייחסי שנקופה זו כך שנוכל לבדוק התנגשות.
5. לסובב את העולם בשלושה המימדים, לסובב אובייקטים בעולם.
6. ציור של כדורים.
7. ציור של מרובע.
8. ציור של חוטים.

# תרשים השליטה השחקן



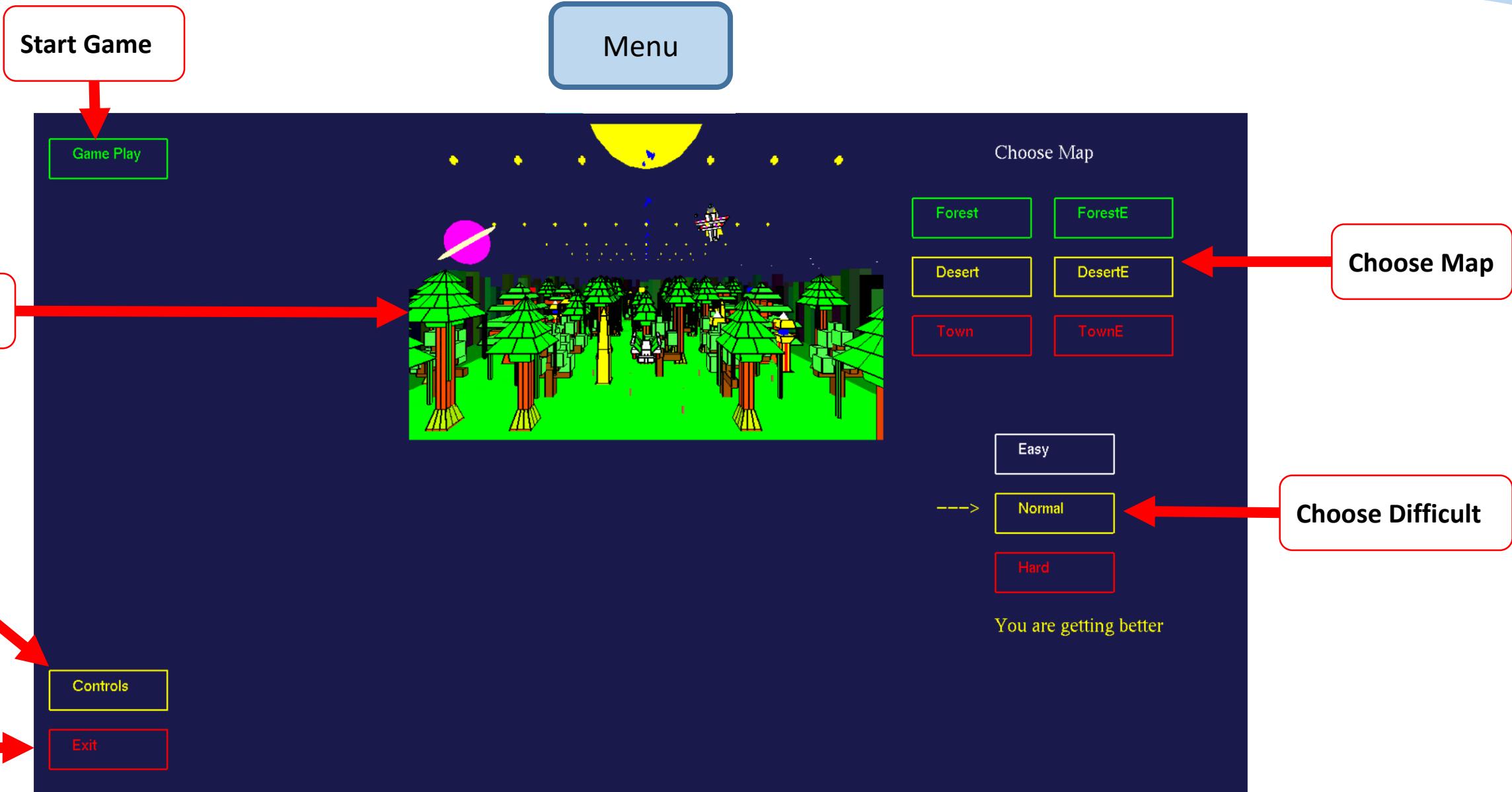
UI

## משחק משתמש



# UI

## משחק משתמש



# UI

## ממשק משתמש

### Control

#### Helicotpers Controls :

F : Fire  
B : Shop  
M : Tank Control Mode  
Z : Change View  
R : Drop Box  
Q : Rotete Left  
E : Rotate Right  
Space : Helicopter Fan Speed Up  
C : Helicopter Fan Speed Down

#### Tanks Control :

A : Attack Base  
P : Attack Helicopter  
P : Protect Helicopter  
O : Protect Base

#### No Tank Yet :

Left Click : On Tank To Choose  
Left Click : SomeWhere To Collect Tanks

#### With Tank :

Left Click : Some Where Send Tank  
Right Click : SomeWhere Holden Tank Left  
Right Click : On Other Tank To Protect It

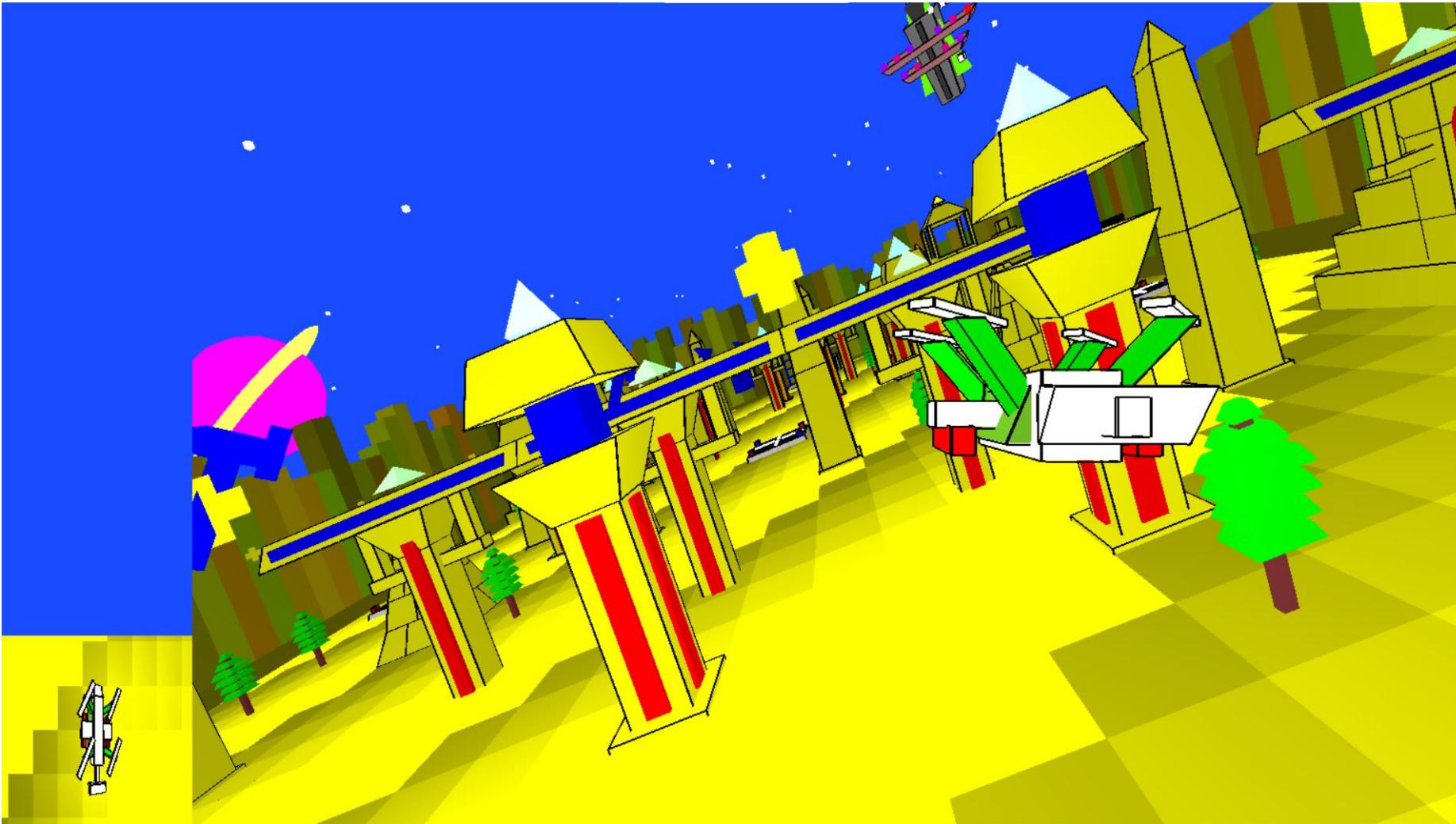
#### Rules :

Player Must Build The Given Build  
Using The Helicotpers To Bring  
Boxes And To Throw Them On His Build

UI

# ממשק משתמש

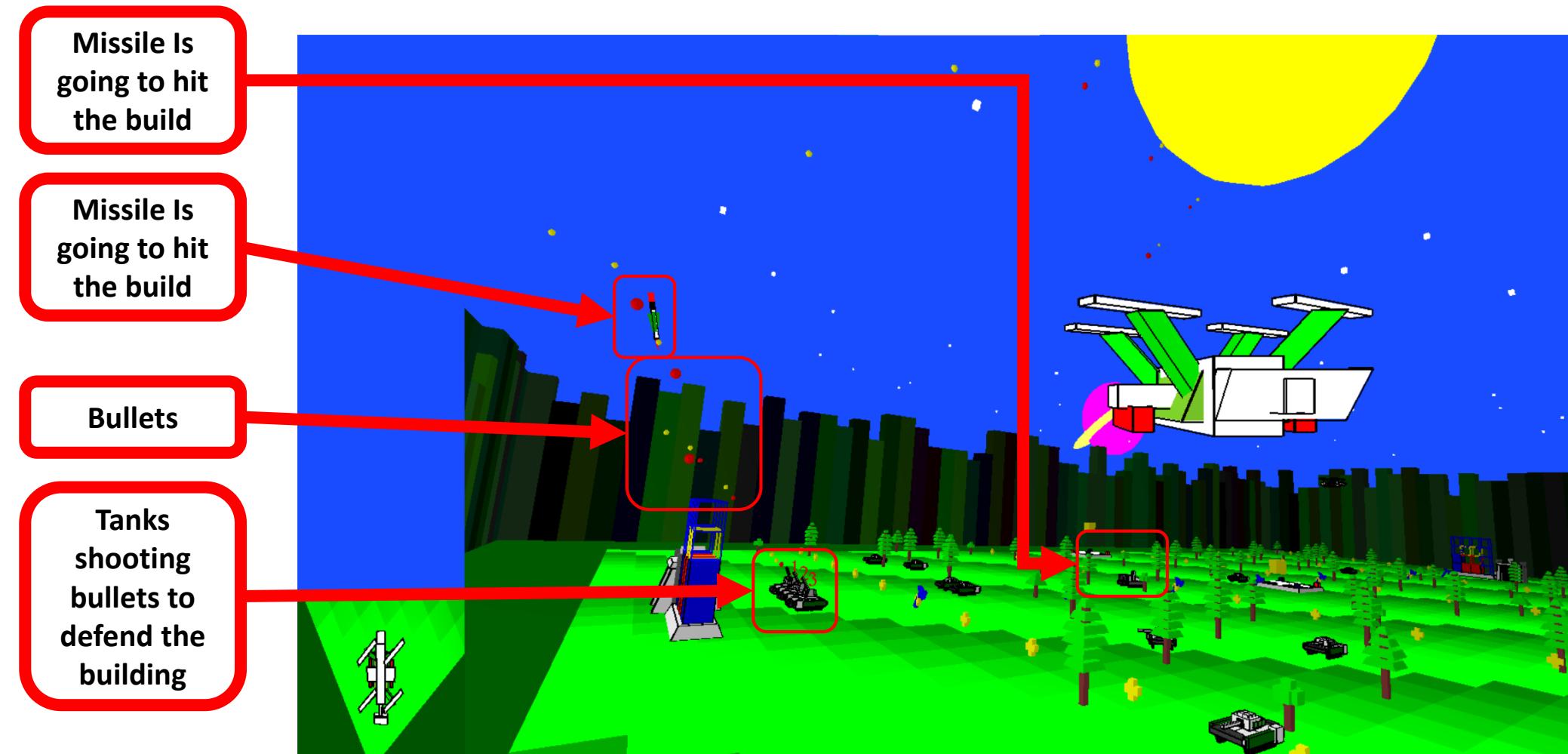
Game



ו

## ממשק משתמש

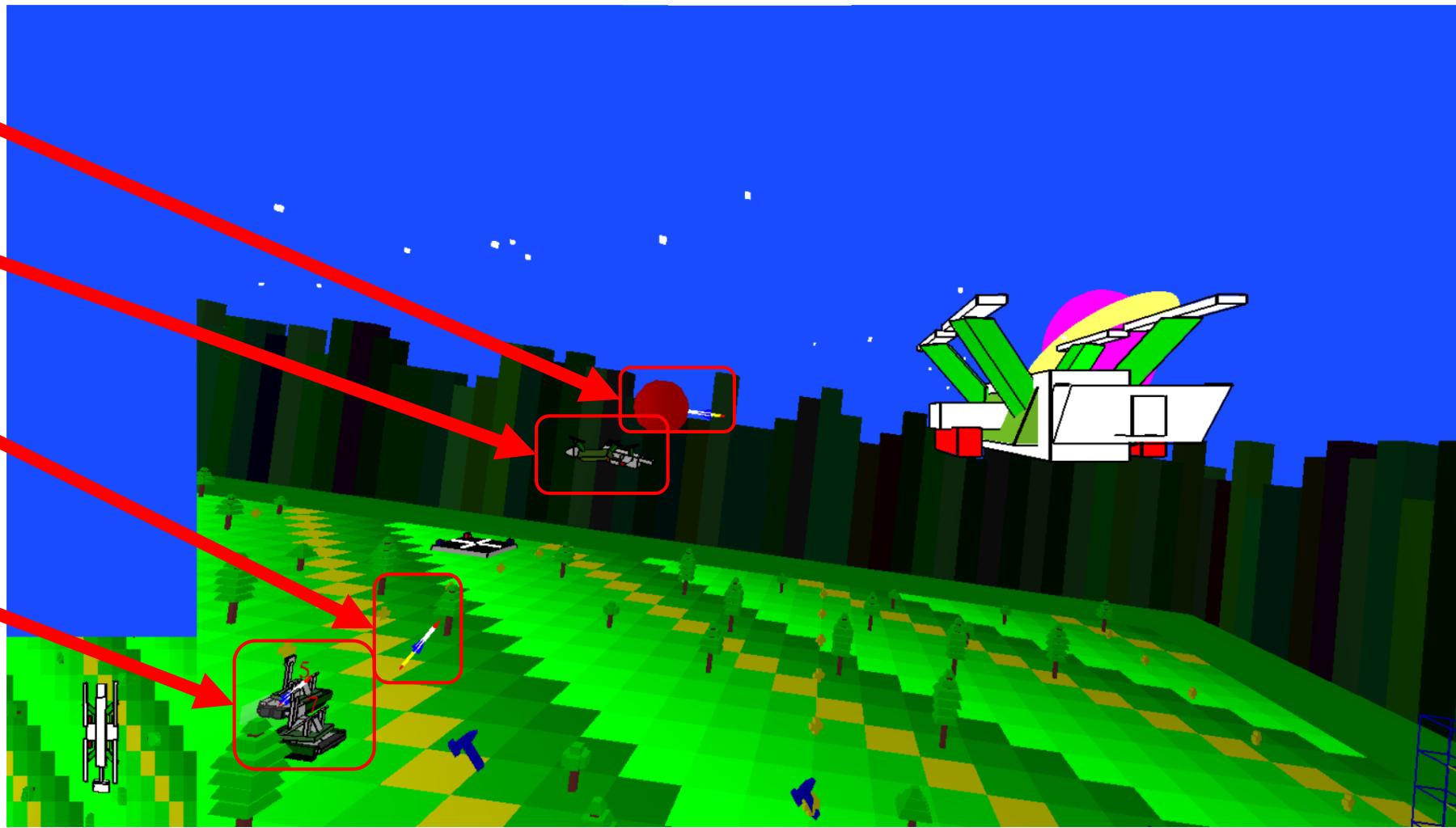
תמונה של: טנק 4 שיורה טיל על הבניין הראשי ויש שלושה טנקים מסוג 2 שמנסים להרוו הטיל לפני שיפגע בבניין.



ו

## ממשק משתמש

תמונה של: טנק 3 שיורה טילים על המסוק של השחקן השני.



# UI ממשק משתמש

Return To Game

Tank Orders

Shop

Back

IDLE

PRT\_BASE

ATK\_BASE

PRT\_HELI

ATK\_HELI

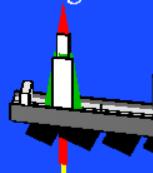
PRT\_TANK

Anti Tanks  
Shoots : High Shells  
Price : 100  
Damage : 50  
Range : 10

Anti Helicopters  
Shoots : High Bullets  
Price : 150  
Damage : 30  
Range : 15

Anti Helicopters  
Shoots : Missiles  
Price : 200  
Damage : 50  
Range : 25

Anti Tanks/Buildings  
Shoots : Missiles  
Price : 200  
Damage : 50  
Range : 50



Buy - 100

Buy - 150

Buy - 200

Buy - 200



# מחלקות

Object	Object Representation
Class D_Vertex	אובייקט מייצג ויקטור בעולם תלת ממד.
Class LocationAble	אובייקט מייצג מיקום.
Class RotationAble	אובייקט מייצג רוטציה.
Class MoveAble	מייצג משק המכיל פונקציה לתנועה.
Class DrawAble	מייצג משק המכיל פונקציה לציור.
Class Node	אובייקט זה מיועד לשמש מרובע ברכפת המשחק ויש בו מצביעים למסלולים הקצרים ביותר לכל מקום במפה.
Class Ground	אובייקט מייצג את המפה שמנוה שמכיל את כל ה-NODES ואחראי להריצת אלגוריתם BFS מכל NODE שכל NODE ידע המסלול הקצר ביותר לכל NODE אחר.
Class Box	אובייקט זה מייצג בוקס תלת ממדי לבניית אובייקטים בצורות שונות, יש לו צבע, מיקום, רוטציה, גודל ותזוזה.
Class BoundAble	אובייקט בוקסים לצייר לבדיקת התנגשות פונקציה גלובלית.

# מחלקות

Object	Object Representation
Class Building	אובייקט מייצג בניין.
Class Bullet	אובייקט מייצג כדורים.
Class Missile	אובייקט מייצג טיל.
Class Helicopter	אובייקט מייצג מסוק שאיתו השחקן ישחק, שהוא גם מכיל BULLETS שיכל לירות אותם.
Class Tank	אובייקט מייצג טנק.
Class Cannon	אובייקט מייצג תותח שיושב על טנק שמכיל MISSILES\BULLETS שיכל לירות אותם.
Class BuildBox	אובייקט זה מייצג הבוקס שהמסוק אמרור להביא אותו ולהרכיב על הבניין, כל זמן מסויים הבוקס אם לא נטפס אז הוא ישנה צבע.
Class Player	אובייקט מייצג שחקן, אובייקט זה יכול את כל הכלים המשחק: מסוק \ טנקים \ בניין ראשי, וגם מכיל את כל התוכניות שהמחשב משחק אותם ומנהל את מחשב.
Class Bang	אובייקטים אלו מייצגים פיצוצים של טילים כדורים.
Class Blow	

# אלגוריתםיקה

ברגע תחילת המשחק ותחול הבניינים של המפה, כל בניין נמצא באיזה מקום שעומד על כמה מרובעים "NODES" המרובעים האילו אנחנו נסים להן STATUS = 0, כי במהלך הרצת את אלגוריתם BFS את האלגוריתם לא יקח אותם בחשבון.

```
void Hold(int x, int z)
{
    if (x < 0 || z < 0 || x >= 30 || z >= 50) return;
    Area[x][z].Status = 0;
}
```

אני פונקציה שمبטלת Node מסוים, עכשו כל ה Nodes נמצאים ב Class Ground אז משם נפעיל את BFS מכל Node.

```
void Set_Neighbours()
{
    int i, j;
    for (i = 0; i < SizeX; i++)
        for (j = 0; j < SizeZ; j++){
            index=indexQ=0;
            FromX = i;
            FromY = j;
            Area[i][j].VisitedFrom[i][j]=&Area[i][j];
            BFS(i, j);
        }
}
```

# אלגוריתם מילוי

אלגוריתם BFS, תמיד מגע לצדדים לפני היזווית כך שייהי את הדרך הקצרה ביותר באופן חמדני.  
באופן חמדני : כך שברור מ Node מסוים בזווית יש מרחק יותר גדול.

```
void BFS(int i,int j)
{
    if (i<0 || j < 0 || i>=SizeX|| j>=SizeZ)return;

    if (i > 0 && Area[i - 1][j].VisitedFrom[FromX][FromY] == NULL && Area[i - 1][j].Status) {
        if (Area[i][j].VisitedFrom[FromX][FromY] != &Area[i - 1][j]) {
            Area[i - 1][j].VisitedFrom[FromX][FromY] = &Area[i][j];
            stackI[index] = i - 1;
            stackJ[index++] = j;
        }
    }

    if (i < SizeX - 1 && Area[i + 1][j].VisitedFrom[FromX][FromY] == NULL && Area[i + 1][j].Status){ ... }
    if (j > 0 && Area[i][j - 1].VisitedFrom[FromX][FromY] == NULL && Area[i][j - 1].Status){ ... }
    if (j < SizeZ - 1 && Area[i][j + 1].VisitedFrom[FromX][FromY] == NULL && Area[i][j + 1].Status){ ... }
    if (i > 0 && j > 0 && Area[i - 1][j - 1].VisitedFrom[FromX][FromY] == NULL && Area[i - 1][j - 1].Status){ ... }
    if (i < SizeX - 1 && j < SizeZ - 1 && Area[i + 1][j + 1].VisitedFrom[FromX][FromY] == NULL && Area[i + 1][j + 1].Status){ ... }
    if (i > 0 && j < SizeZ - 1 && Area[i - 1][j + 1].VisitedFrom[FromX][FromY] == NULL && Area[i - 1][j + 1].Status){ ... }
    if (i < SizeX - 1 && j > 0 && Area[i + 1][j - 1].VisitedFrom[FromX][FromY] == NULL && Area[i + 1][j - 1].Status){ ... }

    while (indexQ!=index) {
        indexQ++;
        BFS(stackI[indexQ-1], stackJ[indexQ-1]);
    }
}
```

אחרי שמנשנו את BFS על כל Nodes, עכשו במשחק כל טנק יכול לדעת המסלול הקצר ביותר לכל ניקודה.

# אלגוריתםיקה

**פונקציה שנמצאת במחלקה Tank שתיקח ניקודה במפה ותמצה המסלול הקצר ביותר לטנק.**

טנק ישתמש בפונקציה זו אם יש לו הוראה ישירה מהשחקן, או אם יש לו מטרה והמטרה זהה למשל : אם טנק X מגן טנק Y , טנק Y צד לאיזה מקום אחר אך טנק X קורה אוטומטית לפונקציה זו.

```
void GetPath(Ground* g, int x, int z)
{
    Node* n;
    int cx, cz;
    cx = (int)Location.x / 2;
    cz = (int)Location.z / 2;
    if (!g->Area[cx][cz].Status)
        return ;
    PathIndex = Status = PathQ = 0;;
    n = &g->Area(cx)[cz];
    if (!g->Area[x][z].Status) return ;
    while (n != &g->Area[x][z])
    {
        if (n == NULL) return ;
        Path[PathIndex++] = n->VisitedFrom[x][z];
        n = n->VisitedFrom[x][z];
    }
    Status = 1;
    return ;
}
```

## חשיבותים מטמטיים

טנק 1 : בודק כל הזמן אם יש טנק של השחקן השני קרוב אליו אם יש אז הוא יקח אותה כי מטרה ומחילה ליאורה עליו כדורים.

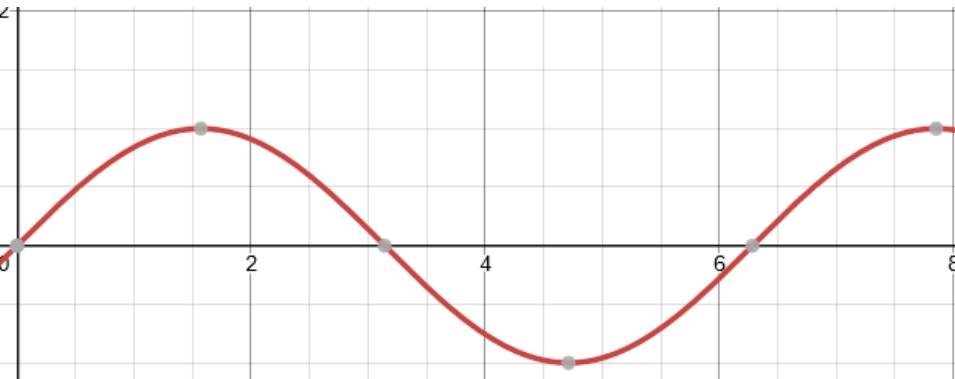
```
if (Distance(&PL1->Tanks[j], &PL2->Tanks[i]) < PL1->Tanks[j].Range)
{
    if (PL1->Tanks[j].Order == IDLE_TANK || PL1->Tanks[j].Order == PROTECT_TANK && !PL1->Tanks[j].Target)
        PL1->Tanks[j].Target = &PL2->Tanks[i];
}
```

עכשו בתוך המחלקה Tank אם יש מטרה אז תחשב את מהירות הכדור בציר X ו Z כך שהכדור תלך לכיוון הנכו, וגם תן לכדור את מיקום הטנק עצמו כך שידע מאיפה להתחל.

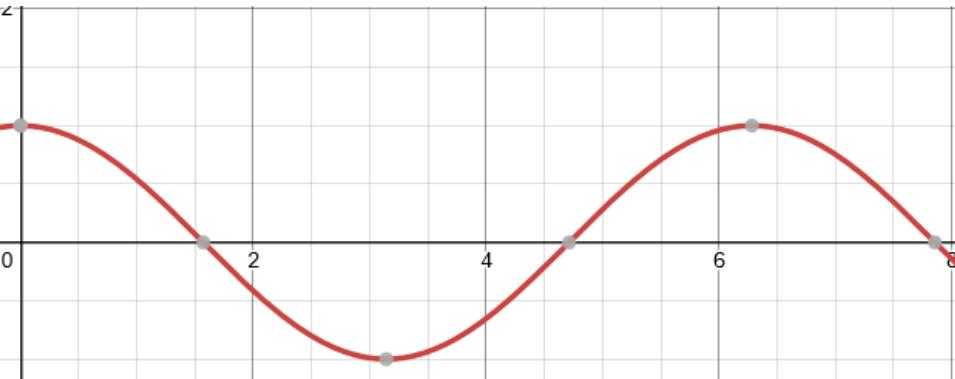
```
if (bullet[0].status) return;
bullet[0].Location.Set_(Location.x, Location.y + 0.5, Location.z);
bullet[0].Speed.x = sin(((cannon.Rotation.y + Rotation.y) * PI) / 180.0) * 0.5;
bullet[0].Speed.z = cos(((cannon.Rotation.y + Rotation.y) * PI) / 180.0) * 0.5;
bullet[0].Status = 1;
Reload = ReloadTime;
```

# חשבונים מטמטיים

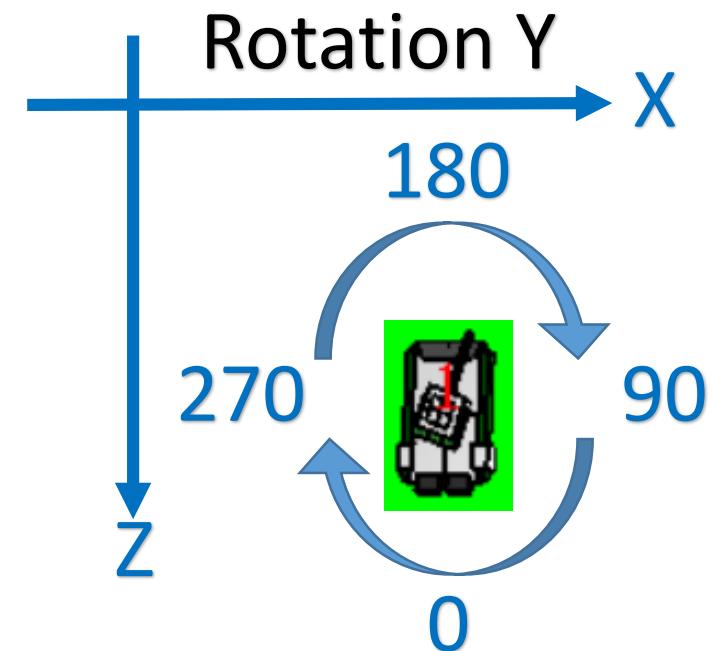
чисוב מהירות הcador זה לפי רוטציית Y של התותח.



```
bullet[0].Speed.x = sin(((cannon.Rotation.y + Rotation.y) * PI) / 180.0) * 0.5;
```



```
bullet[0].Speed.z = cos(((cannon.Rotation.y + Rotation.y) * PI) / 180.0) * 0.5;
```



## חשבונים מטמטיים

טנק 2 : אותו דבר כמו טנק אחד אבל בנוסף נחשב את מהירות של ציר Y, כי טנק זה תוקף מסוק וטילים.  
чисוב מהירות בציר Y זה תלוי ברוטציה X של התותח.

```
if (bullet[BulletIndex].status) return;
bullet[BulletIndex].Location.Set_(Location.x, Location.y + 0.7, Location.z);
bullet[BulletIndex].Speed.x = sin((cannon.Rotation.y + Rotation.y) * PI) / 180.0) * 0.5;
bullet[BulletIndex].Speed.y = sin((cannon.Rotation.x * PI) / 180.0) * -0.5;
bullet[BulletIndex].Speed.z = cos((cannon.Rotation.y + Rotation.y) * PI) / 180.0) * 0.5;
bullet[BulletIndex].Status = 1;
BulletIndex = (BulletIndex + 1) % 5;
Reload = ReloadTime;
```

```
if (cannon.missile_1.Target == NULL) {
    cannon.missile_1.Target = Target;
    cannon.missile_1.Location.x+=Location.x;
    cannon.missile_1.Location.y+=Location.y;
    cannon.missile_1.Location.z+=Location.z;
    cannon.missile_1.Status = 1;
    Reload = ReloadTime;
    return;
}
if (cannon.missile_2.Target == NULL) {
    cannon.missile_2.Target = Target;
    cannon.missile_2.Location.x += Location.x;
    cannon.missile_2.Location.y += Location.y;
    cannon.missile_2.Location.z += Location.z;
    cannon.missile_2.Status = 1;
    Reload = ReloadTime;
    return;
}
```

טנק 3 : בודק כל הזמן אם יש מסוק של השחקן השני קרוב אליו אם יש אז הוא יקח אותה כי מטרה ומתחיל ליאורה אליה טילים.  
ירית טילים: ניתן להטיל את המטרה ואת המיקום של הטנק שיצא ממנו הטיל.

# חשבונים מטמטיים

חישוב רוטציות איני פונקציה שימושת לחישוב רוטציה בציר  $Z$  למקום שבעזרתה נידע:

1. רוטציה הטיל למטרתו
  2. רוטציה הטנק למטרתו
  3. רוטציה התותח למטרתו
- fonction מקבלת  $\text{From} \setminus \text{To} \setminus \text{Add}$

הfonקציה מחשבת את הזווית  $\text{From} \rightarrow \text{To}$  ומוסיפה את  $\text{add}$  לתוכה.

הוספה  $\text{add}$  חסובה במקרים שאנו מחשבים רוטציה של אובייקט שהוא תלוי באובייקט אחר למשל:  
התותח שנמצא על טנק הרוטציה שלו תלויות ברוטציית הטנק אז בזמן חישוב רוטציית התותח אנו נוסיף  $\text{add}$  שהוא ערך השלילי של רוטציית הטנק, כי התותח נכלל ברוטציית הטנק.

```
void RotateUntil_Y(D_Vertex FROM, D_Vertex TO, int add)
{
    if (TO.x > FROM.x){
        if (TO.z > FROM.z)
            Rotation.y = (int)(atan(abs((TO.x) - (FROM.x)) / abs((TO.z) - (FROM.z)))) * 180.0 / PI) + add;
        else
            Rotation.y = 180 - (int)(atan(abs((TO.x) - (FROM.x)) / abs((FROM.z) - (TO.z)))) * 180.0 / PI) + add;
    }
    else{
        if (TO.z > FROM.z)
            Rotation.y = 360 - (int)(atan(abs((FROM.x) - (TO.x)) / abs((TO.z) - (FROM.z)))) * 180.0 / PI) + add;
        else
            Rotation.y = 180 + (int)(atan(abs((FROM.x) - (TO.x)) / abs((FROM.z) - (TO.z)))) * 180.0 / PI) + add;
    }
}
```

# חשבונים מטמטיים

חישוב רוטציות איני פונקציה שמשתמשת לחישוב רוטציה בציר X למקום מסוים שב仄רתה נידע:

1. רוטציה הטיל למטרתו
2. רוטציה הנק למטרתו
3. רוטציה התווחה למטרתו

פונקציה זו מчисבת רוטציה X על 90 מעלות  
אודה סבר כמו הפונקציה שהיתה לפני  
משומשת יחד עם הפונקציה שהיתה לפני  
לחישוב רוטציה X.

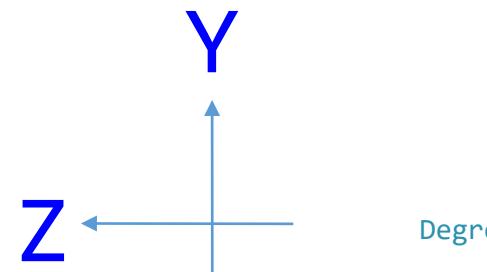
חישוב שתיהיה עד 90 מעלות כי הרוטציה  
שאנחנו מחשבים בו היא אחרי חישוב רוטציה Y  
בר ניקבל רוטציה לכל זווית.

למשל רוטציית הטיל \ רוטציית התווחה.

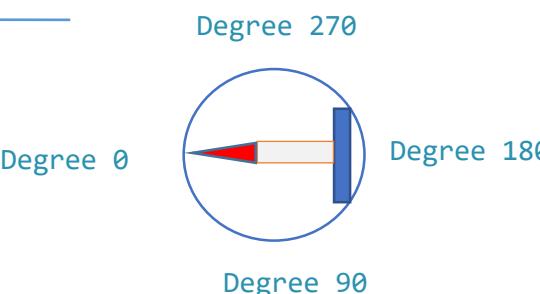
```
void RotateUntil_X_Half(D_Vertex FROM, D_Vertex TO, int add)
{
    float Under;
    if (abs((TO.z) - (FROM.z)) > abs((TO.x) - (FROM.x)))
        Under = abs((TO.z) - (FROM.z));
    else
        Under = abs((TO.x) - (FROM.x));
    if (FROM.z < TO.z){
        if (TO.y > FROM.y)
            Rotation.x = -(int)(atan(abs((TO.y) - (FROM.y)) / Under) * 180.0 / PI) + add;
        else
            Rotation.x = (int)(atan(abs((TO.y) - (FROM.y)) / Under) * 180.0 / PI) + add;
    }
    else{
        if (TO.y > FROM.y)
            Rotation.x = -(int)(atan(abs((TO.y) - (FROM.y)) / Under) * 180.0 / PI) + add;
        else
            Rotation.x = (int)(atan(abs((TO.y) - (FROM.y)) / Under) * 180.0 / PI) + add;
    }
}
```

# חשבונים מטמטיים

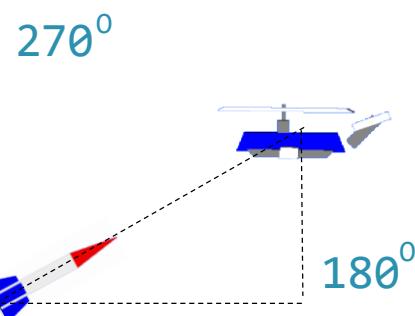
## Missile Rotation



חישוב רוטציה X בין שתי נקודות  
1: נקודת הטיל  
2: נקודת המטרה



$$\text{Tan} = \frac{\text{אורך X}}{\text{אורך Z}}$$



למשל :

$$\text{Missile Degree} = 180 + \text{Tan}(A / B)$$

$$\text{Degree} = 360 - \text{Tan}$$

Target

$$\text{Degree} = 180 + \text{Tan}$$

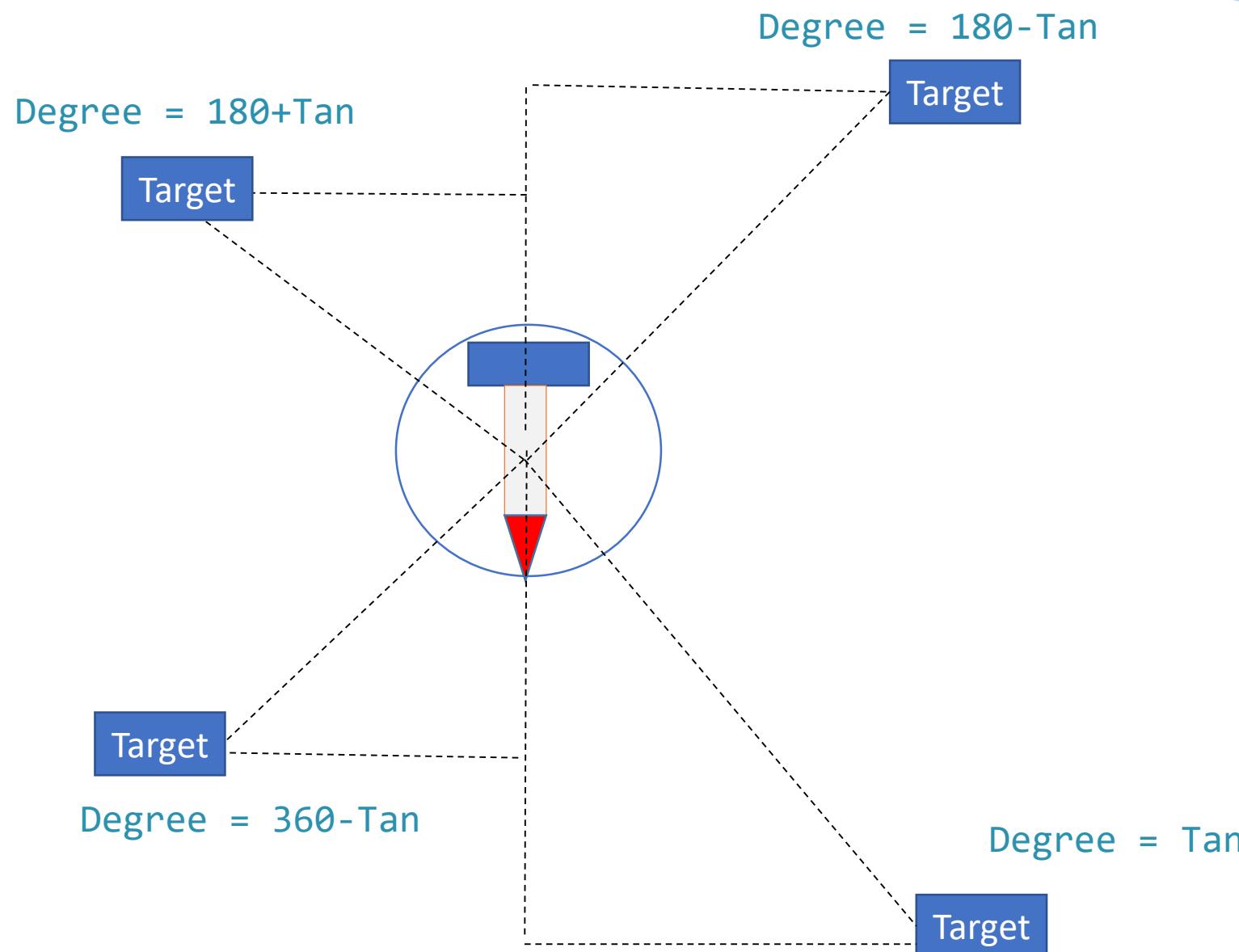
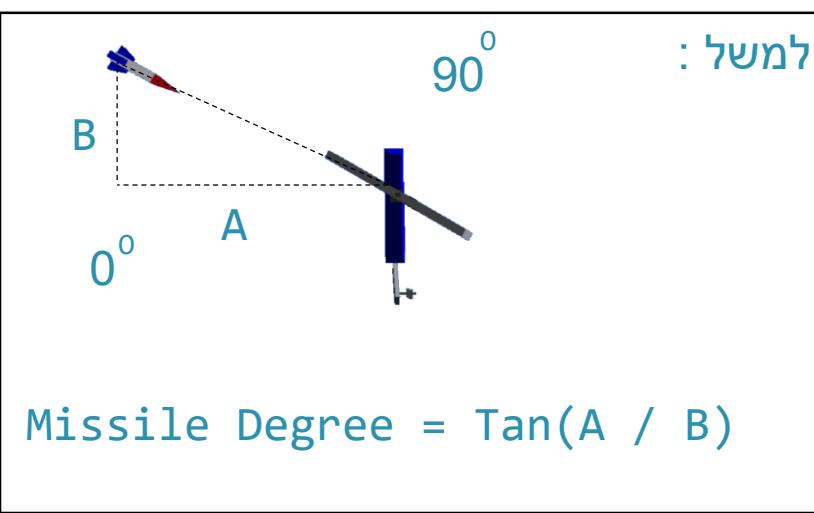
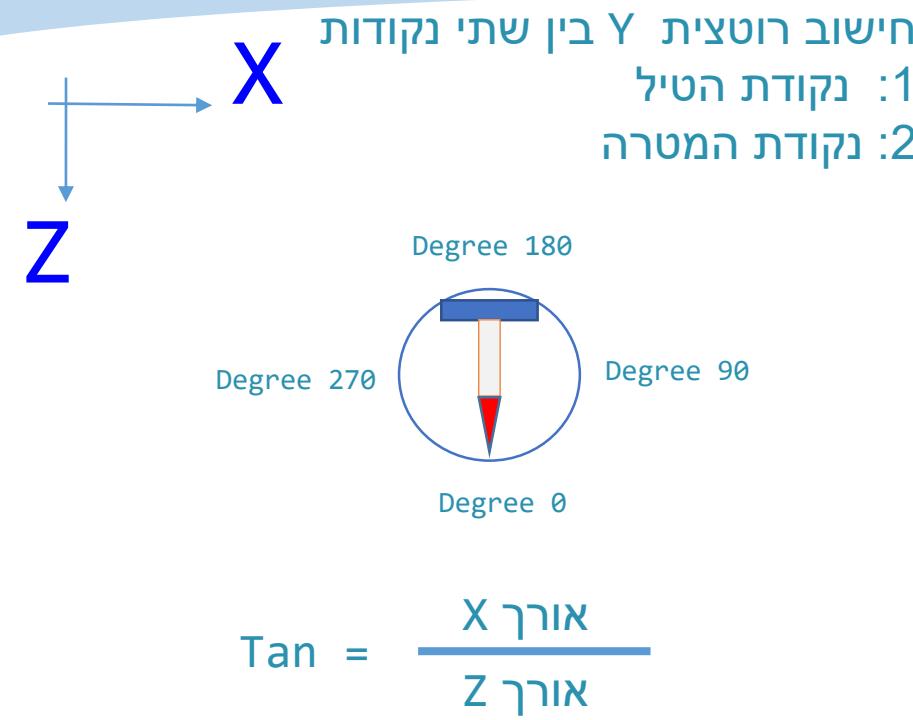
Target

$$\text{Degree} = \text{Tan}$$

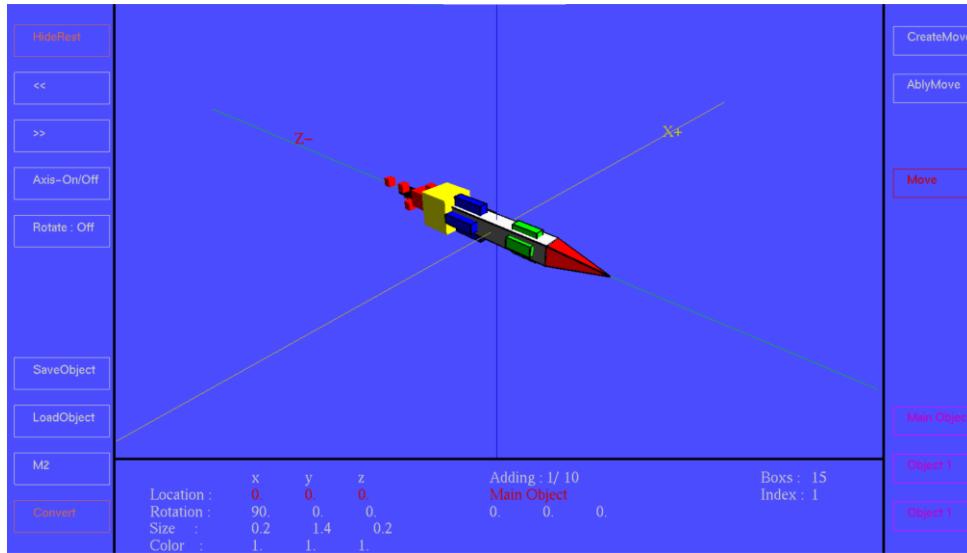
$$\text{Degree} = 180 - \text{Tan}$$

## חישוביים מטמטיים

### Missile Rotation



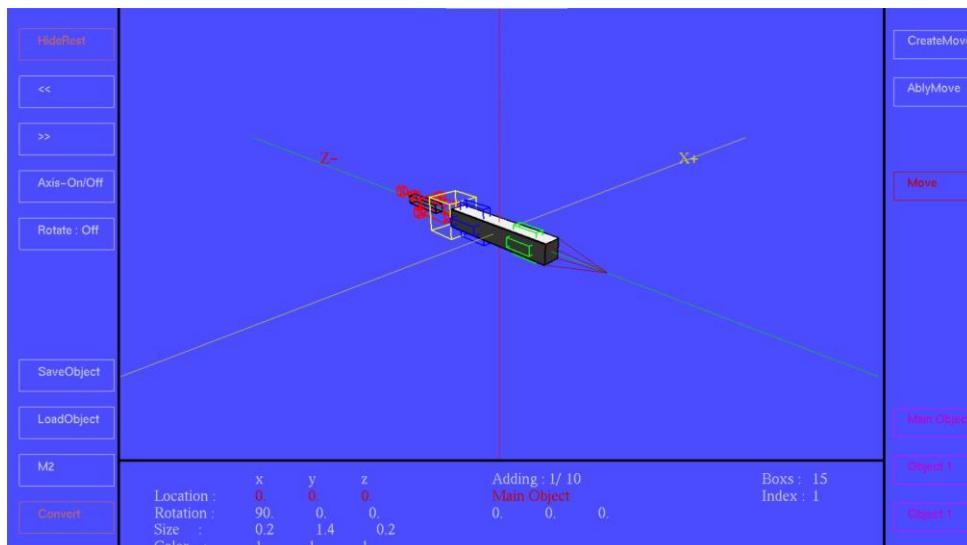
## התנגשות



מיקום של כל Box בכל אובייקט, הוא מיקום ביחס לאובייקט עצמו לביחס העולם של המשחק עצמו, לדוגמה בתמונה יש ציר של טיל ובוקס לבן, שהמקום שלו הוא 0 0 0 ביחס לטיל אבל ביחס לעולם הוא מיקום הטיל.

אך נבנה פונקציה שבודקת אם יש התנגשות בין שני בוקסים.

פונקציה המקבלת שני אובייקטים שהם BoundAble ובודקת התנגשות בין הבוקסים ביניהם:



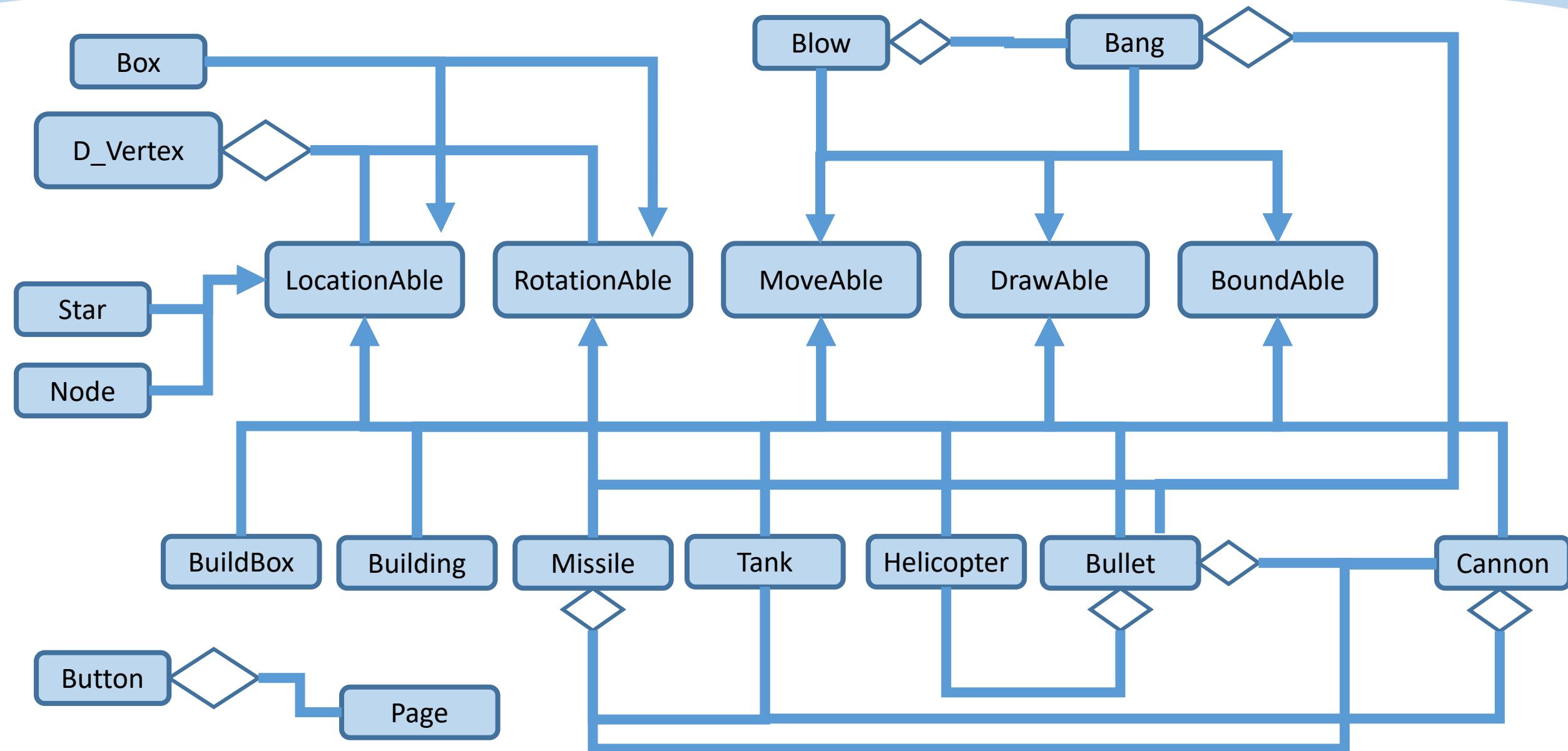
## התנגשות

פונקציה `Box_Bound` מקבלת שניボוקסים, ושני מקומות בהם המיכלים את הボוקסים נוספים לכלボוקס את המיקום של אובייקט המכיל אותו ונבדוק חיתוך ב X Y Z "התנגשות".

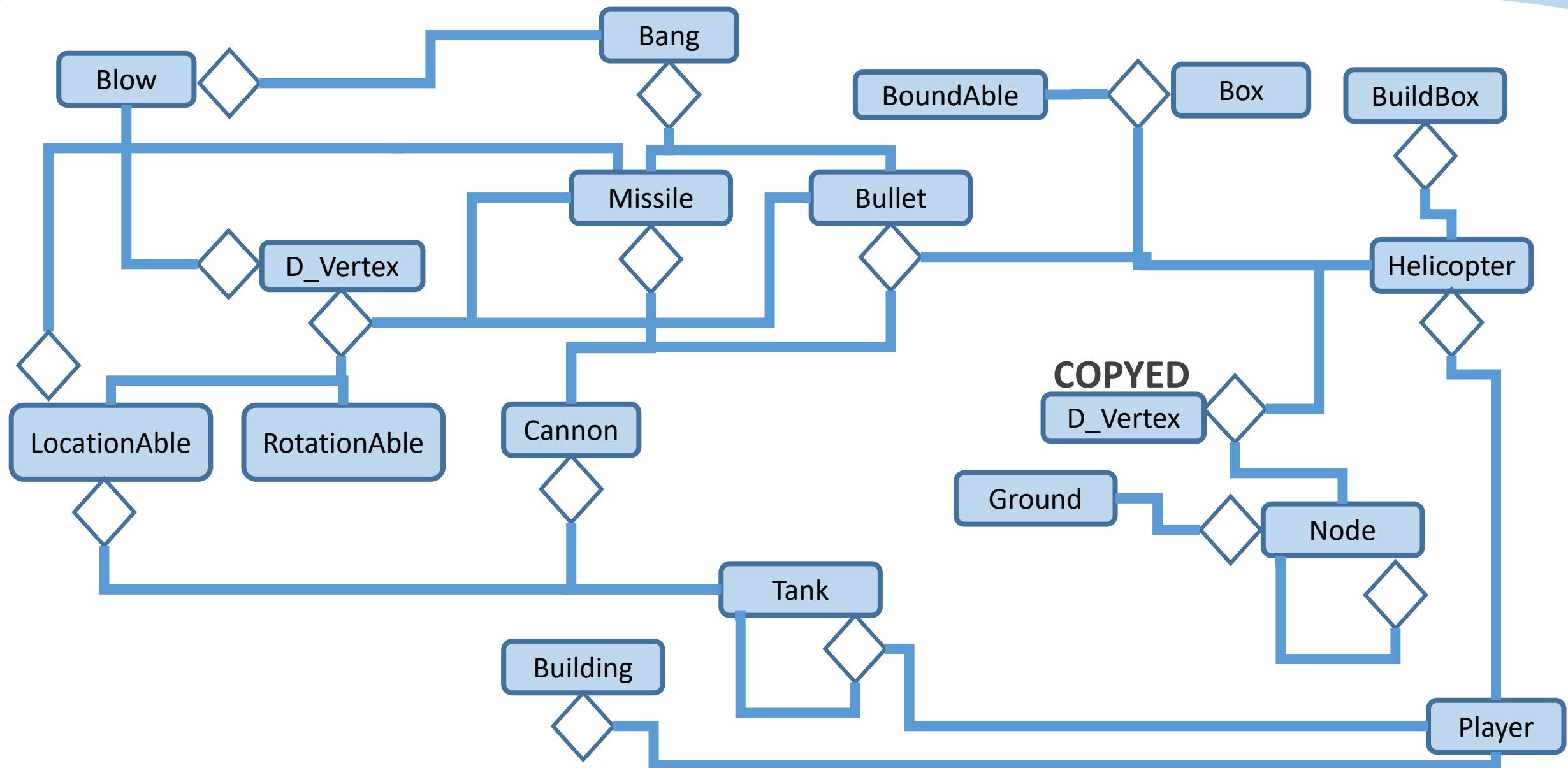
פונקציה `Obj_Bound` מקבלת שני אובייקטים `Boundable` שבתחום מכילים בוקסים ושני מקומות ובודקת אם יש התנגשות בין שני אובייקטים בעזרת פונקציה `Box_Bound`.

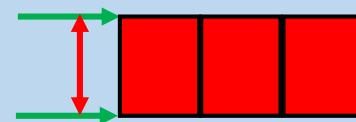
```
int Box_Bound(Box B1, D_Vertex* L1, Box B2, D_Vertex* L2)
{
    B1.Location.Add(L1);
    B2.Location.Add(L2);
    if (B1.Location.x > (B2.Location.x - B2.Size.x / 2 - B1.Size.x / 2) && B1.Location.x < (B2.Location.x + B2.Size.x / 2 + B1.Size.x / 2))
        if (B1.Location.y > (B2.Location.y - B2.Size.y / 2 - B1.Size.y / 2) && B1.Location.y < (B2.Location.y + B2.Size.y / 2 + B1.Size.y / 2))
            if (B1.Location.z > (B2.Location.z - B2.Size.z / 2 - B1.Size.z / 2) && B1.Location.z < (B2.Location.z + B2.Size.z / 2 + B1.Size.z / 2))
                return 1;
    return 0;
}
int Obj_Bound(Boundable* B1, Boundable* B2, D_Vertex* L1, D_Vertex* L2)
{
    if (sqrt(pow(L1->x - L2->x, 2) + pow(L1->y - L2->y, 2) + pow(L1->z - L2->z, 2)) > 10)
        return 0;
    int i, j;
    for (i = 0; i < B1->Box_Count; i++)
        for (j = 0; j < B2->Box_Count; j++)
            if (Box_Bound(B1->box[i], L1, B2->box[j], L2))
                return 1;
    return 0;
}
```

# UML Extends



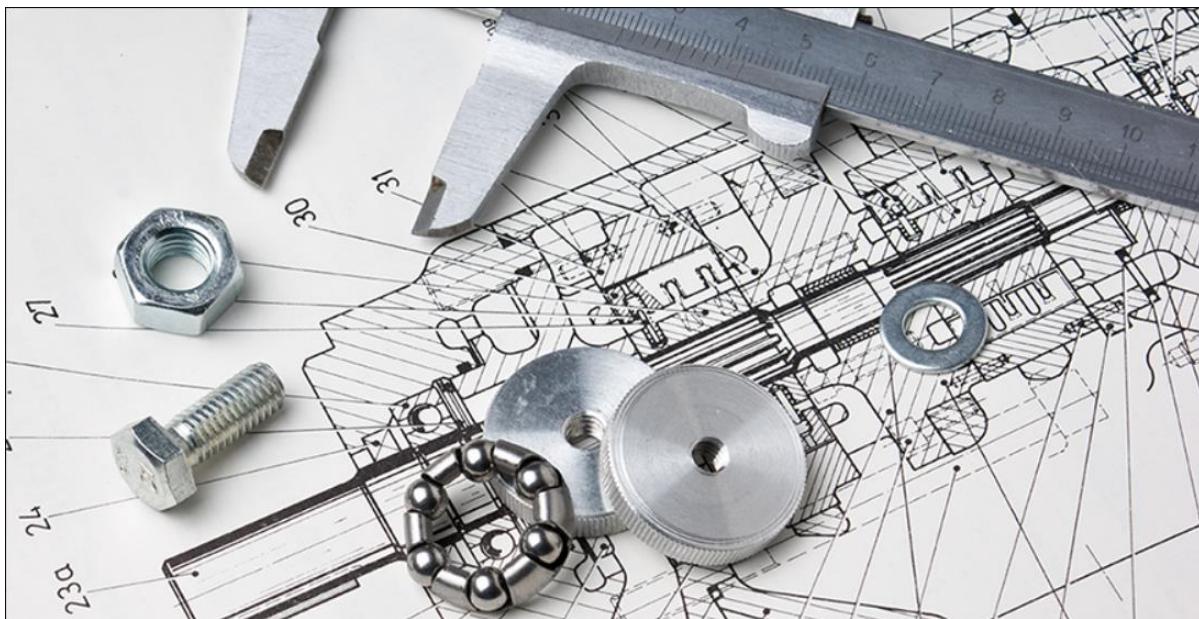
# UML Has A





משום שהאובייקטים בניים מבוקסים קשה מאות ליצת אותם בכתיבת קוד ולאתחל קל בוקס בקוד, זו שיטה אינה יעילה ודורשת המון שורות קוד, لكن טוב יותר לבנות תוכנה שדרקה אפשר ליצור אובייקטים מבוקסים, ובסיום אפשר לשמור אותם בקובץ בינהרי שיש בו את ייצוג הבוקסים של האובייקט.

אותו דבר לגבי מפות, קשה לחשב לבנות מפה דרך כתיבת קוד.

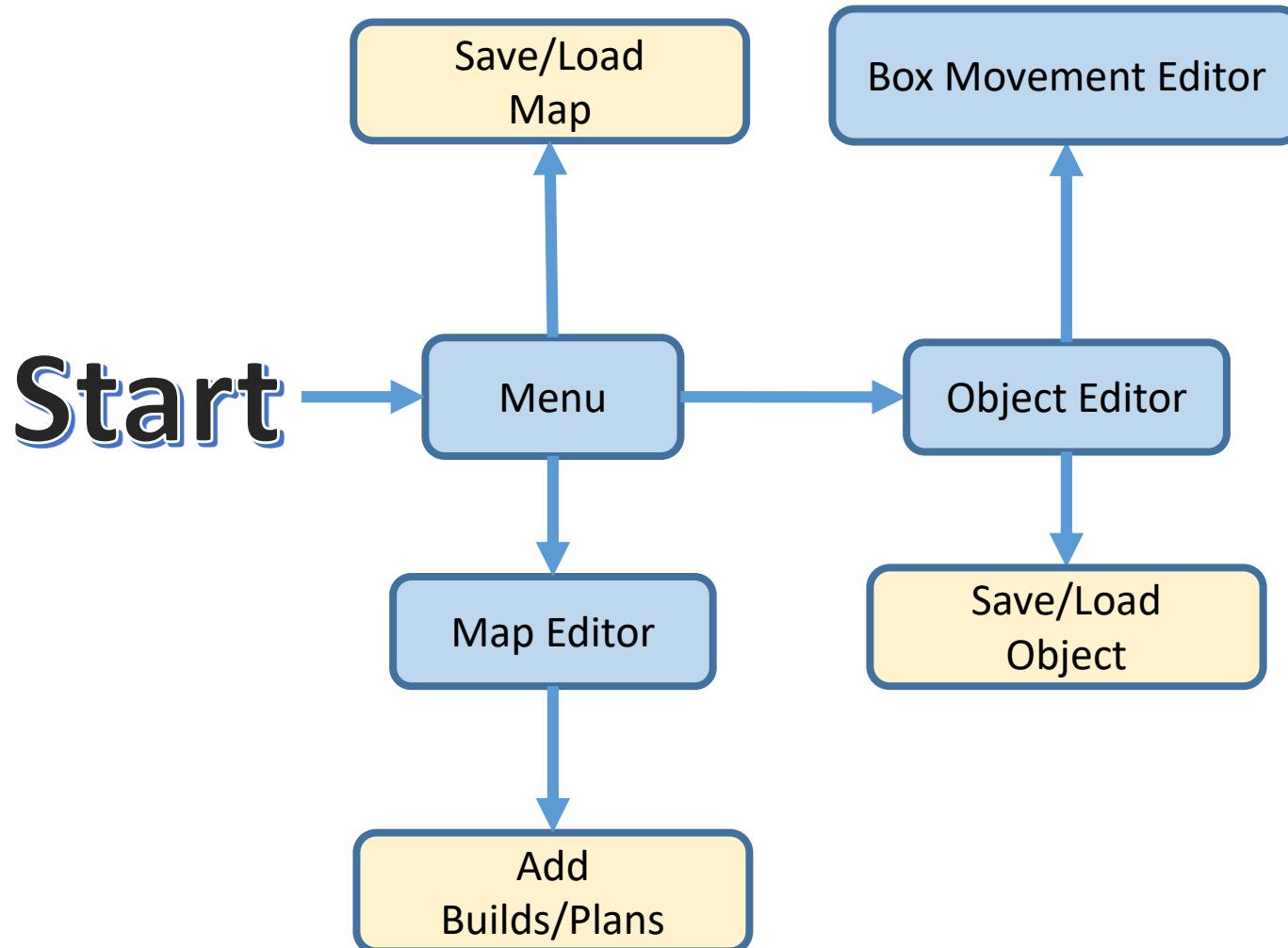


תוכנית יכולה לעשות :

1. לצייר אובייקטים.
2. לשמר אובייקטים.
3. לעדכן אובייקטים קיימים.
4. בניית מפות.
5. בניית תוכנית למחשב.

# Object & Map Editor

או ממשק משתמש



יש מחלקה Box Box שניית ליצר Box עם כלמנה נתוניים : מיקום \ רוטציה \ צבע \ גודל \ שיפוע וגם תזוזה.

יש מחלקה שנקראת Object המכילה מידע של Box, אפשר לחת דרך התוכנה לכל בוקס את הנתוניים שלו ובסוף לשמר את המערך בקובץ בינהרוי שם ה Object בקובץ נפרד כך שנוכל בתוכנה לראות איזה צורות צירנו לפני ומה השם שלהם. אחרי שמירת Object בקובץ אפשר לקרוא אותו בתוך המשחק ולהשתמש בו.

### בנייה מפה :

בזמן זה נקראת כל הציורים שבנוינו קודם ואפשר לקרוא אותם מקבצים ולסדר אותם במפה בעזרת מערך של Object. ונשמר את מיקום כל בנין והשם שלו ומקוםו בוקסים בניו בקובץ שם המפה.

### בנייה תוכנית מחשב :

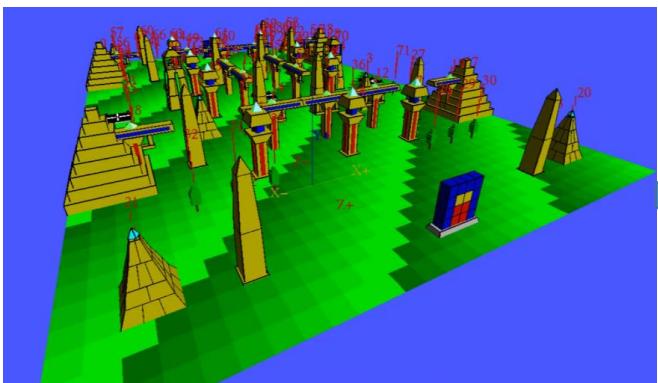
בזמן בניית המפה אפשר לבחור קושא ותוכנית, ולפזר טנקים במפה ולשמור אותם בקבצים.

כך שנוכל לחת מקבצים האילו ונצרף אותם לקבצי המשחק אז נקרא אותם נקל: ציורים \ מפות \ תוכנית מחשב בכל המפה ובכל דרגת קושא.

# Object & Map Editor

שמירת מפה

Build some map



X, Z, TANK\_TYPE

```
Desert_EasyAttackPlan.txt - Notepad
File Edit Format View Help
22 78 4
36 78 4
38 82 1
36 64 1
22 64 1
12 64 1
46 64 1
18 68 2
30 82 3
20 82 1
```

Save map in text files



Desert\_EasyAttackPlan.txt  
Desert\_EasyDefendPlan.txt  
Desert\_EasyMidPlan.txt  
Desert\_HardAttackPlan.txt  
Desert\_HardDefendPlan.txt  
Desert\_HardMidPlan.txt  
Desert\_MidAttackPlan.txt  
Desert\_MidDefendPlan.txt  
Desert\_MidMidPlan.txt

9 files has the bot plan with each difficult Easy/ Normal/ Hard Plan Attack/ Defend/ Medium

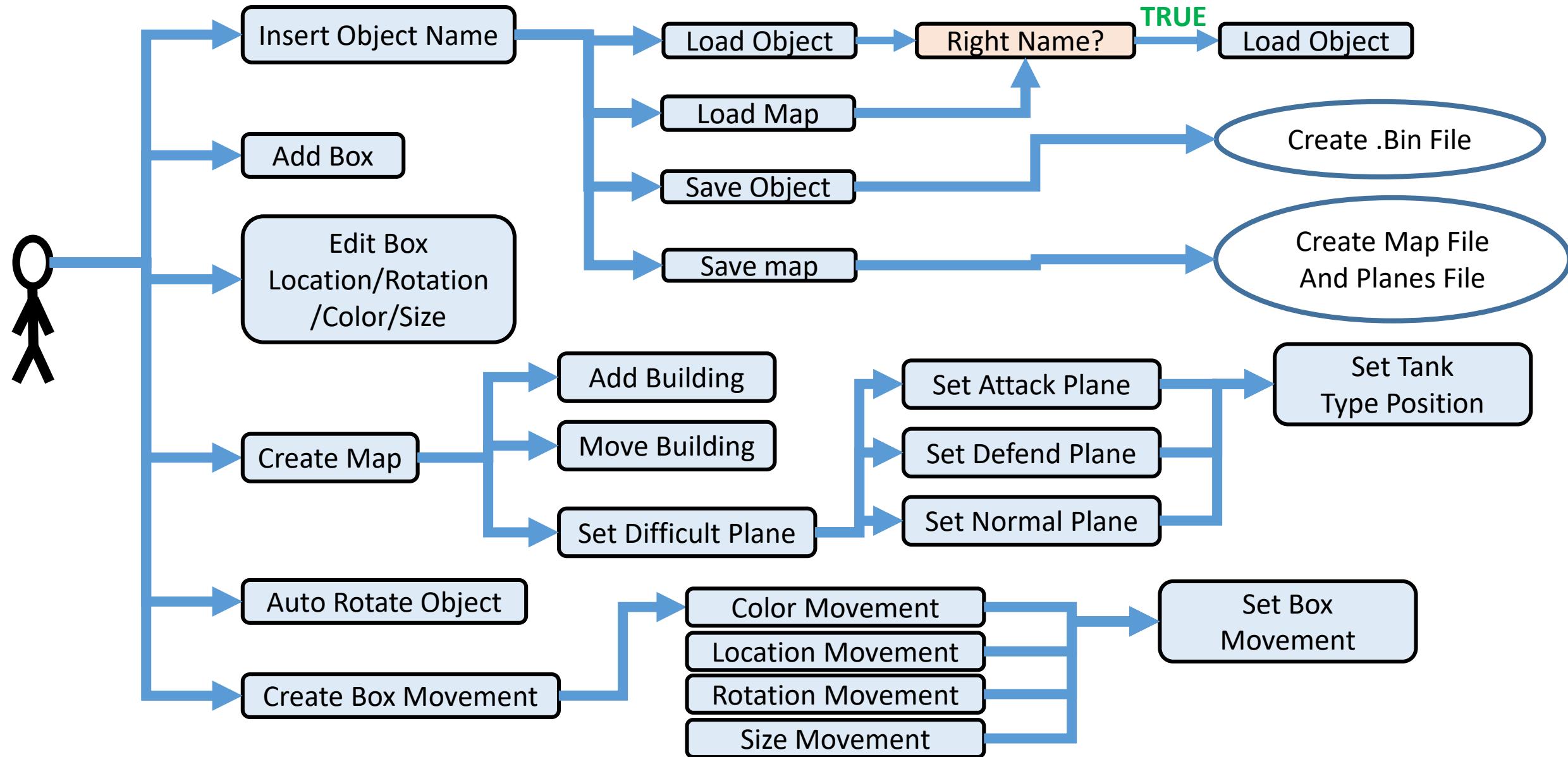
One file has the map buildings location and buildings names X, Y, Z, BOX\_COUNT, BUILD\_NAME

```
Desert.txt - Notepad
File Edit Format View Help
8 3 0 25 Pyramid1
8 3 0 73 Pyramid1
5 15 0 63 Pyramid2
5 43 0 63 Pyramid2
4 46 0 72 PyTower1
4 12 0 72 PyTower1
8 18 0 78 PyTower2
8 40 0 78 PyTower2
8 40 0 42 PyTower2
8 40 0 56 PyTower2
4 24 0 62 PyTower1
4 34 0 62 PyTower1
9 38 0 72 PyTower3F
9 30 0 72 PyTower3F
9 22 0 72 PyTower3F
9 24 0 54 PyTower3R
9 24 0 44 PyTower3R
12 14 0 48 PyTower4
9 6 0 73 PyTower3B
9 52 0 73 PyTower3B
5 53 0 91 Pyramid2
5 5 0 91 Pyramid2
4 12 0 92 PyTower1
4 46 0 92 PyTower1
8 30 0 78 PyTower2
```

For example Difficult : Easy Plan : Attack  
Send the first tank:  
to 22 78 and the tank type is 4

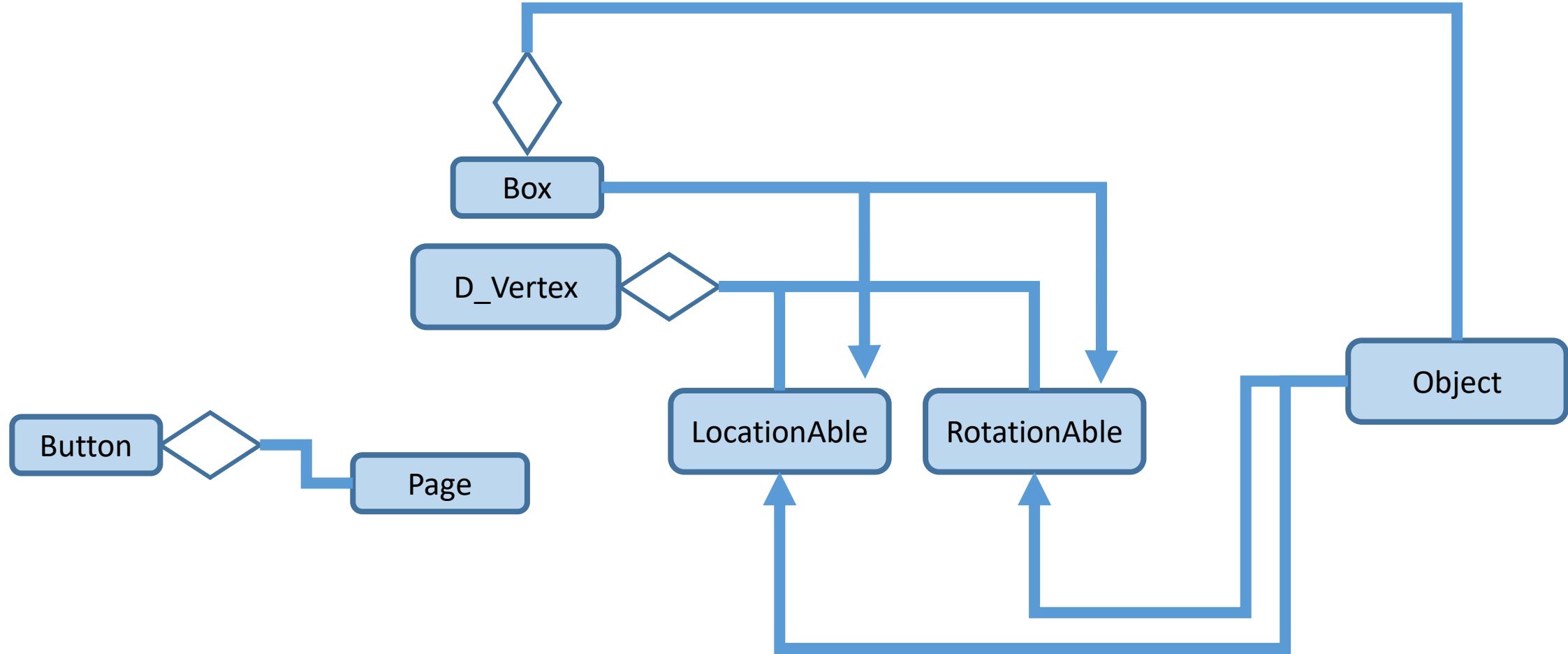
# Object & Map Editor

תרשים השלטונות השחקן



# Object & Map Editor

UML



# Object & Map Editor

אנו ממשק משתמש

Menu

Object Editor

Map Editor

<---- Input Name

SavePlans

Save Map

Load Map

Exit

# Object & Map Editor

עט ממשק משתמש

## Object Editor

HideRest

<<

>>

Axis-On/Off

Rotate : Off

SaveObject

LoadObject

Convert

CreateMove

AblyMove

Move

Main Object

Object 1

Object 1

Location :	x	y	z
Rotation :	0.	0.	0.
Size :	1.	1.	1.
Color :	1.	1.	1.

Adding : 1/ 10

Main Object

0. 0. 0.

Boxes : 1

Index : 0

# Object & Map Editor

עט ממשק משתמש

## Box Movement Editor

--	--	--

Location  
Speed

			N
--	--	--	---

BackMove

--	--	--

Rotation  
Speed

			N
--	--	--	---

ReqMove:OF

--	--	--

Size  
Speed

			N
--	--	--	---

--	--	--

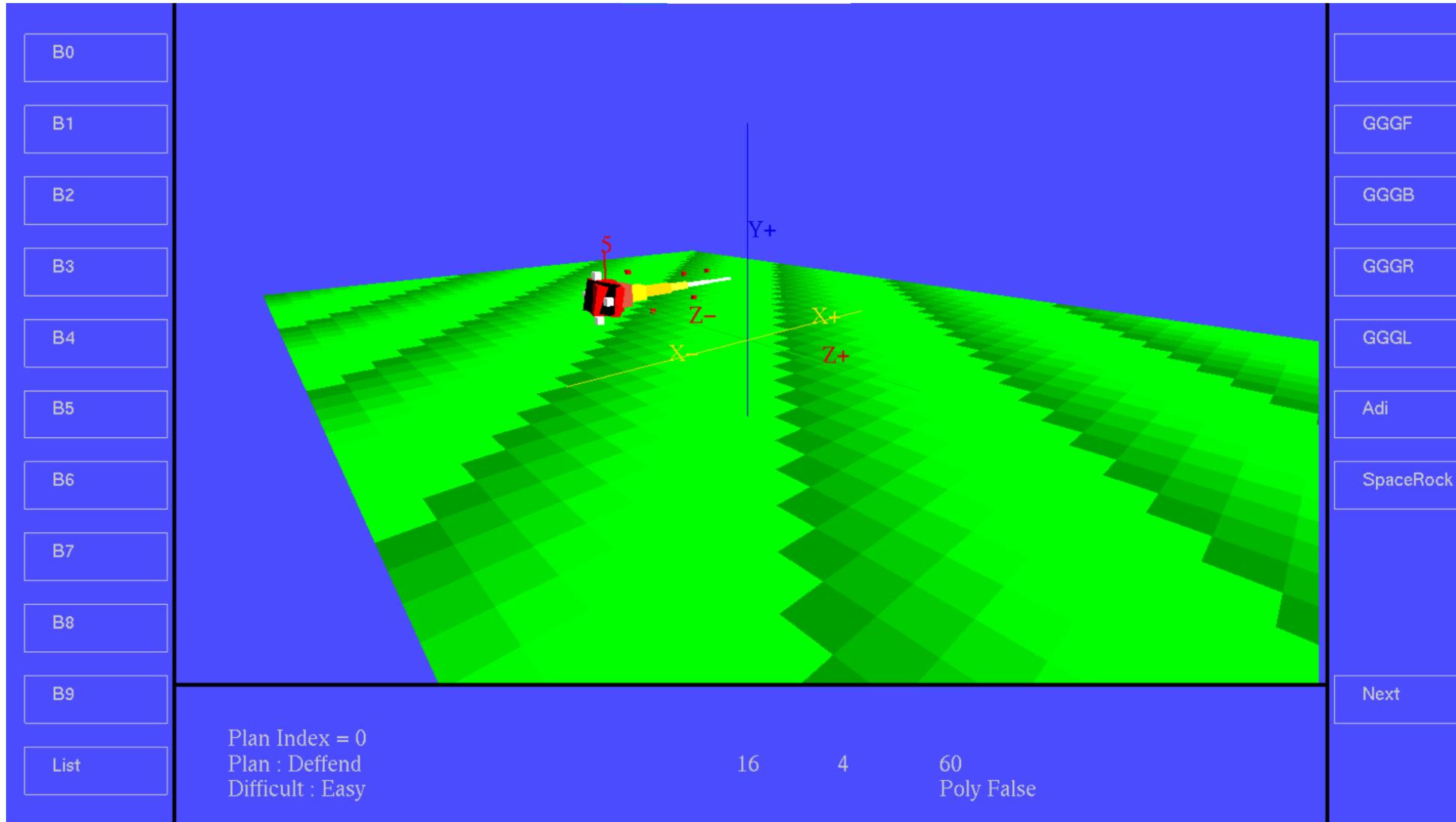
Color  
Speed

			N
--	--	--	---

# Object & Map Editor

עט ממשק משתמש

## Map Editor



Plan Index = 0  
Plan : Deffend  
Difficult : Easy

16      4      60  
Poly False

# Object & Map Editor

עט ממוק משתמש

## Object Editor Page

HideRest

<<

>>

Axis-On/Off

Rotate : Off

SaveObject

LoadObject

SpaceRock

Convert

CreateMove

AblyMove

Move

Main Object

Object 1

Object 1

Working Box Data

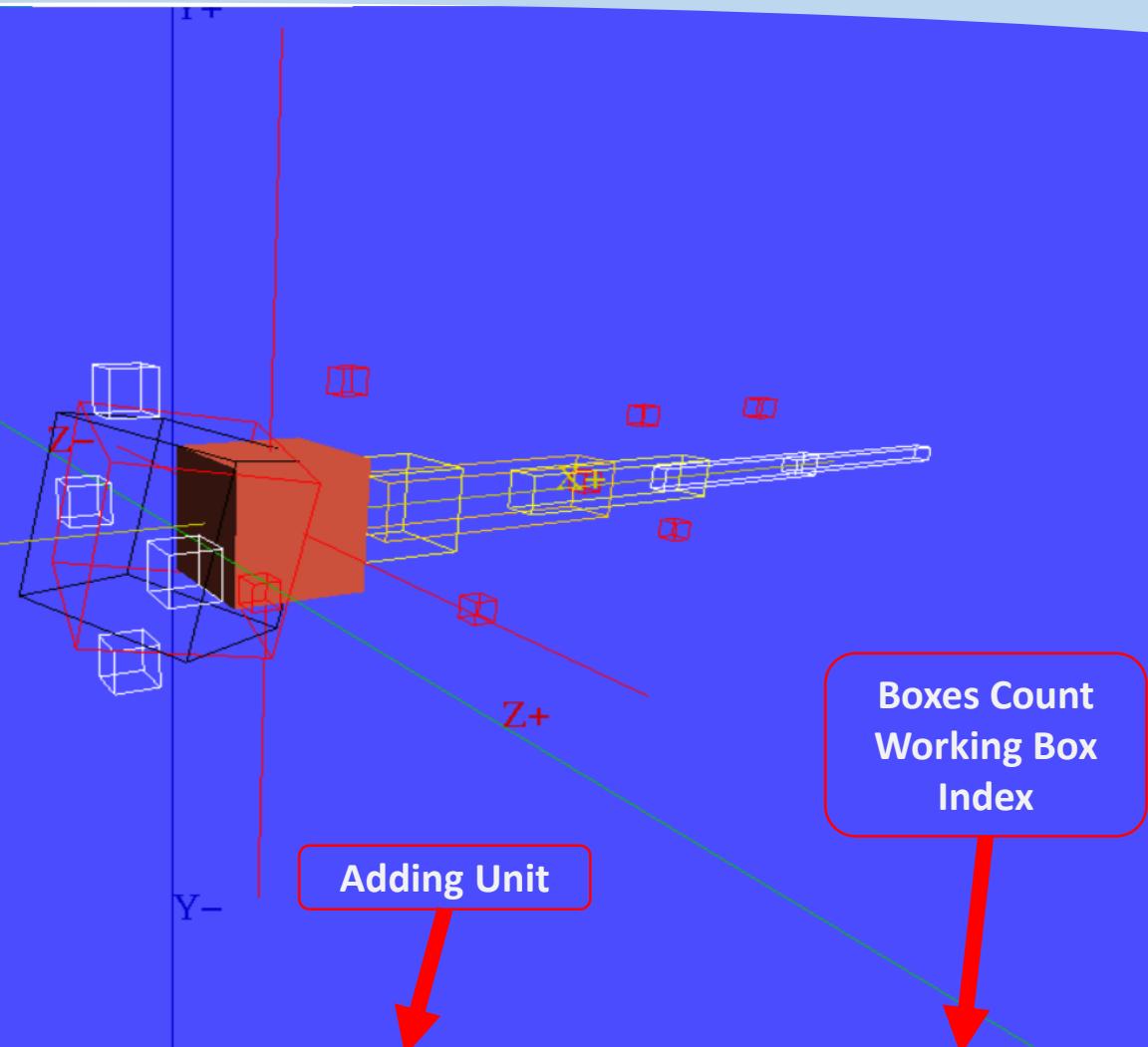
Adding Unit

Boxes Count  
Working Box  
Index

Location : x 1.1 y 0. z 0.  
Rotation : 0. 0. 0.  
Size : 1.5 1.5 1.5  
Color : 1. 0.4 0.3

Adding : 1/ 10  
Main Object  
0. 0. 0.  
Object And Location

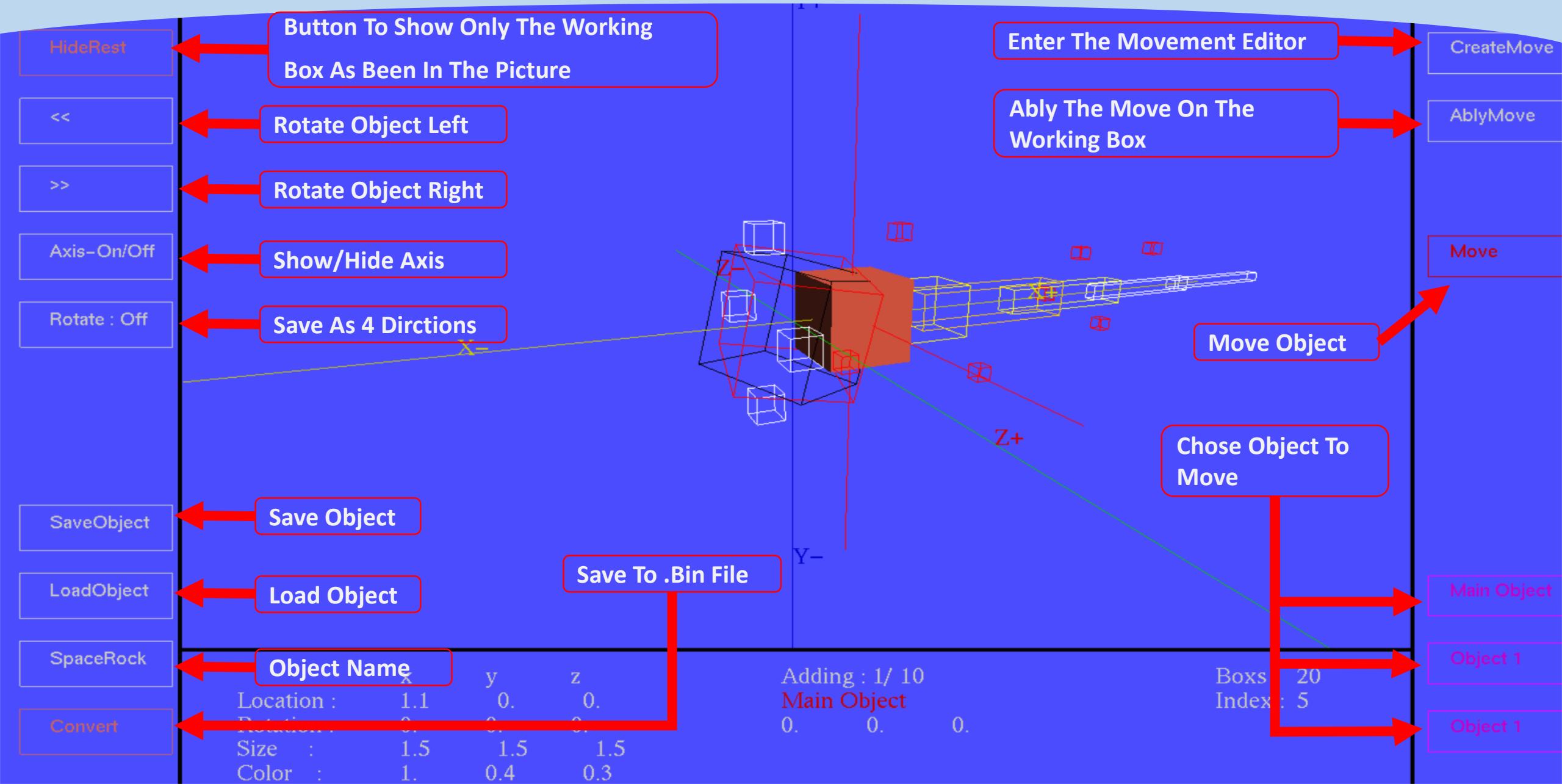
Boxs : 20  
Index : 5



# Object & Map Editor

עט ממשק משתמש UI

## Object Editor Page



# Object & Map Editor

עט ממשק משתמש

## Movement Editor Page

0	0	0	
2	0	0	100

Location  
Speed

**Set Location Move**  
Move x from 0 to 2 with speed 1

Return To  
Object Editor

0	0	0	
0	360	0	050

Rotation  
Speed

**Set Rotation Move**  
Move y from 0 to 360 with speed 5

BackMove

ReqMove:OF

			N

Size  
Speed

			N

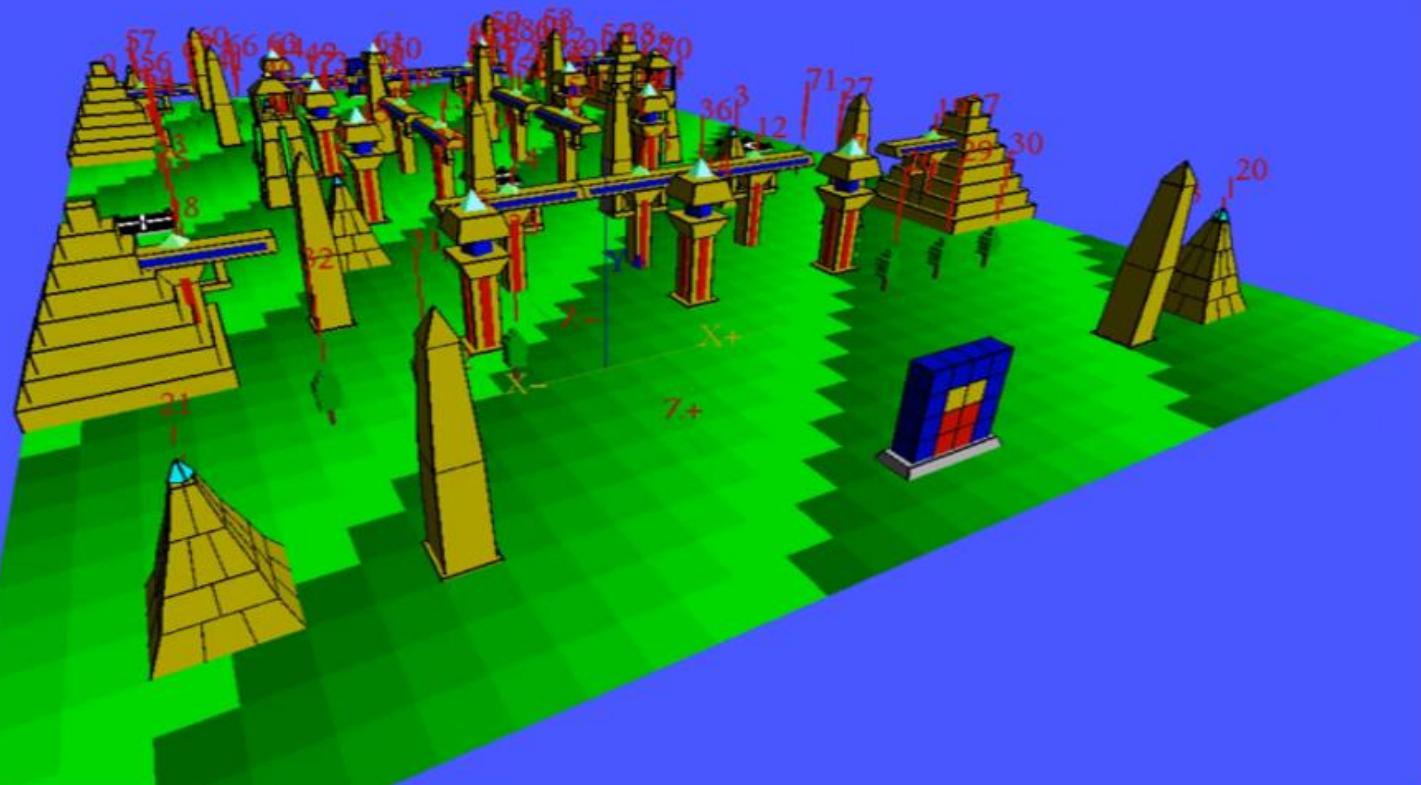
Color  
Speed

**Recursive Move:**  
On : Move From To  
OFF : Move Between From To

Choose Build Number

- B0
- B1
- B2
- B3
- B4
- B5
- B6
- B7
- B8
- B9

List



# UI Management

ב-GL ain כפתורים או דפים או מערכת שמנהל ממשק וו.  
از היצרךתי לבנות מערכת שמנהל את ממשק המשתמש, המערכת הזאת בנוייה בעיקר משני אובייקטים :  
Button \ Page

מחלקה Button : מייצג כפתור על המסך המכיל : מיקום \ מחרוזת \ צבע \ גודל.  
יש לו פונקציה שמקבלת MX MY שהם יהו מיקום לחיצת את העכבר על המסך  
הfonkzia תחזיר אם אלחיצה היא בתוך הכפתור.

```
int Clicked(int Mx, int My)
{
    if (Mx > 770 + (x * 150) - (Side * 150) && Mx < 770 + (x * 150) + (Side * 150))
        if (My > 430 + (y * -150) - (Side * 50) && My < 430 + (y * -150) + (Side * 50))
            return 1;
    return 0;
}
```

ציות הכפתור הוא פשוט 4 חוטים, והדפסת את המחרוזת בתוכם.

```
void Draw_Button()
{
    Color.Color_On();
    glBegin(GL_LINE_LOOP);
    glVertex3f(x + Side, y + Side / 3, -5);
    glVertex3f(x - Side, y + Side / 3, -5);
    glVertex3f(x - Side, y - Side / 3, -5);
    glVertex3f(x + Side, y - Side / 3, -5);
    glEnd();
    strPointer = str;
    glColor3f(Color.x, Color.y, Color.z);
    glRasterPos3f(x - 0.3, y, -5);
    do glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *strPointer); while (*(++strPointer));
```

# UI Management

מחלקה Page : מייצג דף המכיל : מערך מצבעי כפתורים \ מצבע לפונקציה \ מצבע לדף שהוא ה-"Previous Page". אז בזמן אתחול את המשחק או את התוכנה בזמן אתחול הפטורים אלו נשלח כל כפתור לאיזה Page הוא שיירCCR שכל Page יוכל את הפטורים המתאים, בנוסף נשלח לדף את הפונקציה שהוא אמרור דרך לצייר את ממשך המשחק המשתמש.

בסוף Page יוכל כתוב את כתובות הדף שלפניו.

```
class Page
{
public:
    int BtnsSize;
    Button* Btns[250];
    Page* Back;
    void (*PagePrint)();
    void SetPrintFuction(void (*PagePrint)())
    {
        this->PagePrint = PagePrint;
    }
    void AddButton(Button* btn)
    {
        if (BtnsSize >= 250) return;
        Btns[BtnsSize++] = btn;
    }
    void Show()
    {
        PagePrint();
    }
};
```

צייר דף : פשוט נצייר את כל הפטורים ונפעיל את הפונקציה שהדף יוכל את הכתובת שלו באמצעות פונקציה Show().

# UI Management

בטור המשחק או התוכנה אנו נאתחל כמה דפים, ומצביע "Page \*Current\_Page" שהוא מצביע על דף ראשי, ותמיד אנחנו מפעילים לצייר את (show->Current, `Current->show`, אם צריכים להחלים דף אנו פשוט מצביעים על הדף המתאים).

## בדיקה לחיצת כפתור:

אנו משתמשים בכמה defines בשביל לנוחות בדיקה שכל Define הוא ביטוי בוליאני שבודק אם דף נכון.

```
#define MAIN_PAGE (Current_Page == &MainPage)
#define MAP_PAGE (Current_Page == &MapPage)
#define OBJECT_PAGE (Current_Page == &ObjectPage)
#define MOVEMENT_PAGE (Current_Page == &MovementPage)
#define EXIT_PAGE (Current_Page == &ExitPage)
```

```
if (MAIN_PAGE)
{
    if (BuildMap.Clicked(x, y)) {
        Current_Page = &MapPage;
        return;
    }
    if (BuildObject.Clicked(x, y)) { ... }
    if (LoadMap.Clicked(x, y)) { ... }
    if (SaveMap.Clicked(x, y)) { ... }
    if (ExitButton.Clicked(x, y)) { ... }
    if (TypeObject.Clicked(x, y)) { ... }
    if (SavePlans.Clicked(x, y)) { ... }
}
```

שבעזרתם אזמן לחיצת עכבר אנו יכול לבדוק באיזה רף אנו נמצאים, כך שאוכל לבדוק לחיצת הcptורים המתאים.

# Build & Destroy

BUILD & DESTROY

