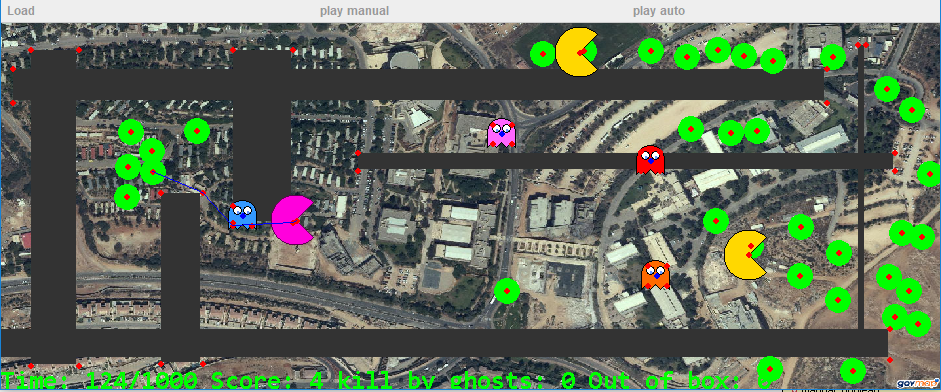
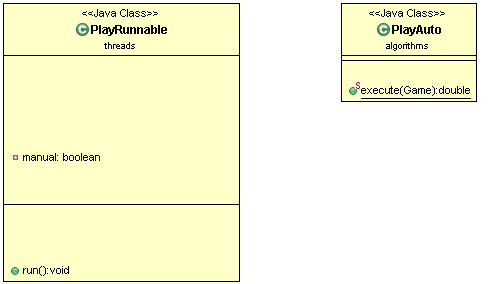
מערכת זו אני יסביר ב2 חלקים. הראשון היא עיצוב המחלקות והשני הוא האלגוריתם.



# עיצוב המחלקות של המערכת.



המשחק רץ על תרד בשם PlayRunnable אשר מקבל כפרמטר משתנה מסוג Boolean אשר אומר לו האם לשחק אוטומטית או על ידי העכבר. תהליכון(thread) זה נוצר על ידי MyFrame ונקרא כאשר לוחצים על הכפתור play manual או PlayAuto. למידע נוסף יש את הדיאגרמה של הפרויקט. במידה והמשתנה הוא false התרד יריץ אלגוריתם אשר מקבל את כל הנתונים במשחק ויחזיר זווית אליה השחקן צריך ללכת. במידה וערך של משתנה זה הוא true הזווית תהיה פשוט מהנקודה בא העכבר נמצא. האלגוריתם יחשב בפונקציה סטטית execute אשר נמצאת במחלקת PlayAuto. במחלקה זו יהיו כול הפונקציות הקשורות לאלגוריתם והיא תקרא גם לפונקציות העושות Dijkstra. פונקציות אלה נלקחו מהאתר baeldung.com.נכתב על ידי אביב וקסלר



אני הוספתי בעצמי עוד 2 משתנים למחלקה Node בהם אני ישתמש. הראשון הוא flag האומר אם הנקודה היא נקודה שאני צריך להגיע עליה או לא. השני זה הנקודה במיקום על המפה.

# האלגוריתם



באלגוריתם זה אני הולך לשלוח את Me אל הפרי הכי קרוב או אל הפקמן הכי קרוב. האלגוריתם מחשב מרחק של המסלול הכולל ולא רק דיסטנס בין 2 נקודות. הדרך בא Me יודע איך לעקוף את הקירות היא בכך שהוא עובר דרך צמתים המובילים אותו אל הפרי. חישוב המסלול דרך צמתים יתבצע על ידי דייקסטרה. אני יכתוב את האלגוריתם המייצר את הצמתים. לאחר מכן אני יעבור על כול הצמתים עליהם אני צריך להגיע וישלח את Me לצומת הכי קרוב. בקוד ה-java התבספו שיפורים שאינם מפועים בפסאודו קוד שהם: עקיפת הרוחות, ציור נקודות על המפה, ציור קווים על המפה והתחשבות במקרה קצה בוא קופסא חורגת מגבולות הלוח. עקיפת הרוחות תתבצע על ידי ריבוע אותו אני יצייר על הרוח. נכתב על ידי אביב וקסלר

pseudo code:

execute(game):

1. nodes[\*] = makeNodes(game)
2. nodes[\*] = calculateShortestPathFromSource(nodes[\*],nodes[0]) //nodes[0] is Me.
3. double distance = ∞
4. double angle;
5. for(i = 0 to size(Nodes))
   1. if(nodes[i].flag)
      1. if(nodes[i].distance < distance )
         1. distance = nodes[i].distance
         2. angle = azimute(game.me, firstPoint(Nodes[i]) )
6. return angle

מחזיר את הנקודה הכי קרובה ליה הפקמן צריך ללכת.

firstPoint(node)

1. if(node. getShortestPath().size() > 1)
   1. return node. getShortestPath().get(1).point
2. else
   1. return node.point

בונה את הצמתים. אני יקח אתגר לצייר את הצמתים על המפה כדי שאני יוכל לבדוק שהפונקציה עובדת. הציור על המפה במשחק יקרה רק על ידי הפעלת ציאט.

makeNodes(game):

1. nodes[\*]; // build dynamic array
2. buildNodes(nodes[\*],game)
3. removeBoxNodes(nodes[\*],game)
4. destinationNodes(nodes [\*],game)

בנה את כול הצמתים גם אם הם על קיר. נוריד אותם בפונקציה אחר-כך.

buildNodes(nodes[\*],game):

1. i = 0
2. nodes[i] = new Node(“me”, false, game.me) //(name,flag,point)
3. //builde node to any fruit

while(fruit = game. iteratorFruit())

* 1. nodes[i++] = new Node(“fruit”, true, fruit)

1. //builde node to any packmen

while(packmen = game. iteratorPackmen ())

* 1. nodes[i++] = new Node(“packmen”, true, packmen)

1. //builde 4 nodes to any box

while(box = game.iteratorBox ())

* 1. Point = (left - 1, top - 1)
  2. nodes[i++] = new Node(“box”, false, Point)
  3. Point = (right + 1, top - 1)
  4. nodes[i++] = new Node(“box”, false, Point)
  5. Point = (right + 1, buttom + 1)
  6. nodes[i++] = new Node(“box”, false, Point)
  7. Point = (left - 1, buttom + 1)
  8. nodes[i++] = new Node(“box”, false, Point)

מסיר צמתים הנמצאים בתוך הקירות כי Me לא יכול להגיע אליהם.

removeBoxNodes(nodes[\*],game):

1. for(i = 0 to size(nodes))
   1. while(box = game.iteratorBox ())
      1. if( box.left < nodes[i].x < box.right &&

box.top < nodes[i].y < box. buttom)

* + - 1. remove(nodes[i--])

מחבר את הצמתים

destinationNodes(nodes[\*],game):

1. for(i = 0 to size(nodes))
   1. for(j = 0 to size(nodes))
      1. if(!isBoxMiddle(nodes[i].point, nodes[j].point,game))
         1. nodes[i]. addDestination(nodes[j], distance(nodes[i], nodes[j]))))

בודק האם יש קופסא באמצע. עובד לפי גאומטריה אנליטית. יש לשים לב שבקורדינציות פולריות ציר x וציר y מתחלפים.

isBoxMiddle(pA,pB,game):

