

TUGAS BESAR 1
STRATEGI ALGORITMA

Pemanfaatan Algoritma Greedy dalam pembuatan bot permainan Diamonds



Disusun Oleh:

Evelyn Yosiana (13522083)
Fedrianz Dharma (13522090)
Steven Tjhia (13522103)

Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

DAFTAR ISI

DAFTAR ISI.....	2
DAFTAR GAMBAR.....	3
BAB 1.....	1
BAB 2.....	5
2.1. Algoritma Greedy secara umum.....	5
2.2. Alur Menjalankan Program.....	6
2.3. Alur pengembangan bot.....	6
BAB 3.....	8
3.1. Alternatif Solusi.....	8
3.2. Strategi Terpilih.....	13
BAB 4.....	15
4.1. Pseudocode.....	15
4.2. Pengujian.....	27
4.3 Hasil dan Pembahasan.....	49
BAB 5.....	50
5.1. Kesimpulan.....	50
5.2. Saran.....	50
Lampiran.....	51
Daftar Pustaka.....	52

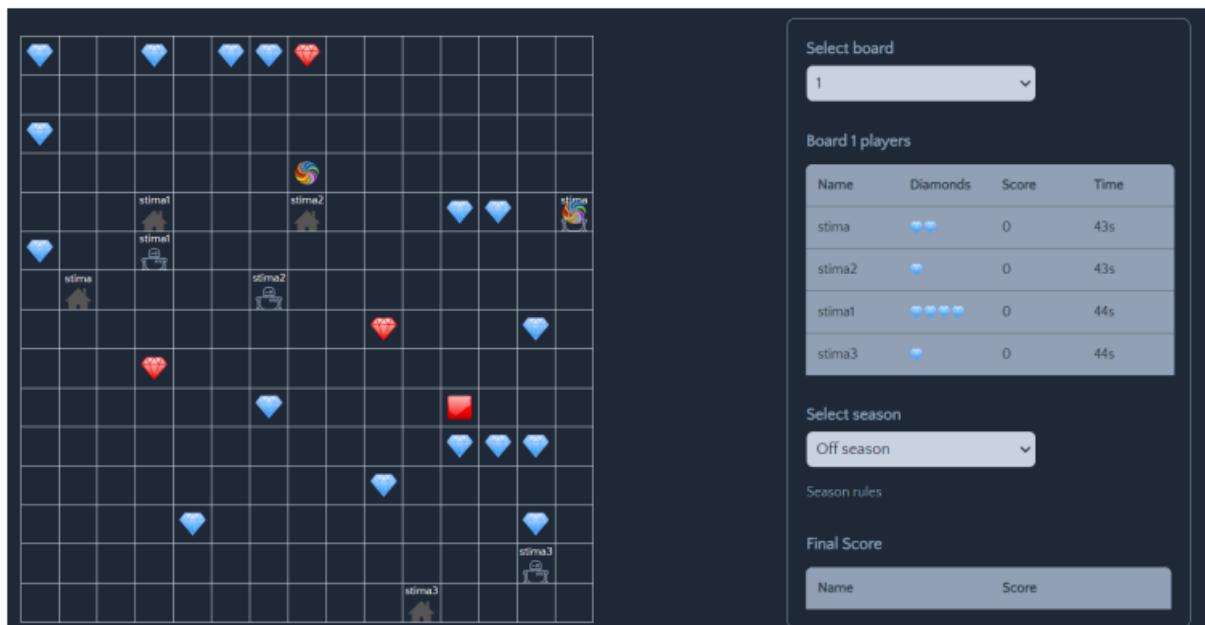
DAFTAR GAMBAR

Gambar 1.1 Tampilan Permainan.....	1
Gambar 1.2 Diamonds.....	2
Gambar 1.3 Red Button.....	3
Gambar 1.4 Teleporters.....	3
Gambar 1.5 Bots and Bases.....	3
Gambar 2.3.1. Lokasi base.py.....	6
Gambar 2.3.2. Class BaseLogic di base.py.....	7
Gambar 2.3.3. Modifikasi next_move() pada FirstToTheKeyFirstToTheEggBot.py yang telah meng-inherit class BaseLogic.....	7

BAB 1

Deskripsi Tugas

Diamonds merupakan suatu *programming challenge* yang mempertandingkan bot yang anda buat dengan bot dari para pemain lainnya. Setiap pemain akan memiliki sebuah bot dimana tujuan dari bot ini adalah mengumpulkan diamond sebanyak-banyaknya. Cara mengumpulkan diamond tersebut tidak akan sesederhana itu, tentunya akan terdapat berbagai rintangan yang akan membuat permainan ini menjadi lebih seru dan kompleks. Untuk memenangkan pertandingan, setiap pemain harus mengimplementasikan strategi tertentu pada masing-masing bot-nya. Penjelasan lebih lanjut mengenai aturan permainan akan dijelaskan di bawah.



Gambar 1.1 Tampilan Permainan

Pada tugas pertama Strategi Algoritma ini, mahasiswa diminta untuk membuat sebuah bot yang nantinya akan dipertandingkan satu sama lain. Tentunya mahasiswa harus menggunakan strategi *greedy* dalam membuat bot ini.

Program permainan Diamonds terdiri atas:

1. Game engine, yang secara umum berisi:
 - a. Kode backend permainan, yang berisi logic permainan secara keseluruhan serta API yang disediakan untuk berkomunikasi dengan frontend dan program bot.
 - b. Kode frontend permainan, yang berfungsi untuk memvisualisasikan permainan.
2. Bot starter pack, yang secara umum berisi:
 - a. Program untuk memanggil API yang tersedia pada backend.
 - b. Program bot logic (bagian ini yang akan kalian implementasikan dengan algoritma greedy untuk bot kelompok kalian).
 - c. Program utama (main) dan utilitas lainnya Untuk mengimplementasikan algoritma pada bot tersebut, mahasiswa dapat menggunakan game engine dan membuat bot dari bot starter pack yang telah tersedia pada pranala berikut.
 - Game engine :
<https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0>
 - Bot starter pack :
<https://github.com/haziqam/tubes1-IF2211-bot-starter-pack/releases/tag/v1.0.1>

Komponen-komponen dari permainan Diamonds antara lain:

1. Diamonds



Gambar 1.2 Diamonds

Untuk memenangkan pertandingan, kita harus mengumpulkan diamond ini sebanyak-banyaknya dengan melewati/melangkahinya. Terdapat 2 jenis diamond yaitu diamond biru dan diamond merah. Diamond merah bernilai 2 poin, sedangkan yang biru bernilai 1 poin. Diamond akan di-regenerate secara berkala dan rasio antara diamond merah dan biru ini akan berubah setiap regeneration.

2. Red Button/Diamond Button



Gambar 1.3 Red Button

Ketika *red button* ini dilewati/dilangkahi, semua diamond (termasuk red diamond) akan di-generate kembali pada board dengan posisi acak. Posisi *red button* ini juga akan berubah secara acak jika *red button* ini dilangkahi.

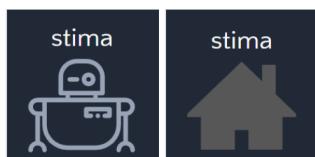
3. Teleporters



Gambar 1.4 Teleporters

Terdapat 2 teleporter yang saling terhubung satu sama lain. Jika bot melewati sebuah teleporter maka bot akan berpindah menuju posisi teleporter yang lain.

4. Bots and Bases



Gambar 1.5 Bots and Bases

Pada game ini kita akan menggerakkan bot untuk mendapatkan diamond sebanyak banyaknya. Semua bot memiliki sebuah Base dimana Base ini akan digunakan untuk menyimpan diamond yang sedang dibawa. Apabila diamond disimpan ke base, score bot akan bertambah senilai diamond yang dibawa dan inventory (akan dijelaskan di bawah) bot menjadi kosong.

5. Inventory

Name	Diamonds	Score	Time
stima	♥♥	0	43s
stima2	♥	0	43s
stima1	♥♥♥♥	0	44s
stima3	♥	0	44s

Gambar 1.6 Inventory

Bot memiliki inventory yang berfungsi sebagai tempat penyimpanan sementara diamond yang telah diambil. Inventory ini memiliki kapasitas maksimum sehingga IF2211 Strategi Algoritma – Tugas Besar 1 3 sewaktu waktu bisa penuh. Agar inventory ini tidak penuh, bot bisa menyimpan isi inventory ke base agar inventory bisa kosong kembali.

Untuk mengetahui flow dari game ini, berikut ini adalah cara kerja permainan Diamonds.

1. Pertama, setiap pemain (bot) akan ditempatkan pada board secara random. Masing-masing bot akan mempunyai home base, serta memiliki score dan inventory awal bernilai nol.
2. Setiap bot diberikan waktu untuk bergerak, waktu yang diberikan semua sama untuk setiap pemain.
3. Objektif utama bot adalah mengambil diamond-diamond yang ada di peta sebanyak-banyaknya. Seperti yang sudah disebutkan di atas, diamond yang berwarna merah memiliki 2 poin dan diamond yang berwarna biru memiliki 1 poin.
4. Setiap bot juga memiliki sebuah inventory, dimana inventory berfungsi sebagai tempat penyimpanan sementara diamond yang telah diambil. Inventory ini sewaktu-waktu bisa penuh, maka dari itu bot harus segera kembali ke home base.
5. Apabila bot menuju ke posisi home base, score bot akan bertambah senilai diamond yang tersimpan pada inventory dan inventory bot akan menjadi kosong kembali.
6. Usahakan agar bot anda tidak bertemu dengan bot lawan. Jika bot A menimpa posisi bot B, bot B akan dikirim ke home base dan semua diamond pada inventory bot B akan hilang, diambil masuk ke inventory bot A (istilahnya tackle).
7. Selain itu, terdapat beberapa fitur tambahan seperti teleporter dan *red button* yang dapat digunakan apabila anda menuju posisi objek tersebut.
8. Apabila waktu seluruh bot telah berakhir, maka permainan berakhir. Score masing-masing pemain akan ditampilkan pada tabel Final Score di sisi kanan layar.

BAB 2

Landasan Teori

2.1. Algoritma Greedy secara umum

Algoritma *greedy* adalah algoritma yang menyelesaikan suatu persoalan dengan memecah masalah dan menyelesaiannya secara *step-by-step*. Pada tiap langkah, program dengan algoritma *greedy* akan mengambil pilihan terbaik yang akan memberikan hasil optimal pada saat itu dengan prinsip *take what you can get now*, tanpa memikirkan konsekuensinya ke depan, bahkan jika sebenarnya ada pilihan yang lebih menguntungkan di akhir namun kurang terlihat menguntungkan di awal. Dalam Algoritma Greedy tidak diperbolehkan melakukan backtracking sehingga pemrogram diharapkan untuk memilih solusi yang paling optimal pada langkah tersebut (local optimum). Hal ini dilakukan dengan harapan solusi-solusi optimum lokal tersebut akan menghasilkan solusi yang optimum untuk keseluruhan persoalan (global optimum).

Elemen-elemen dalam algoritma *greedy* antara lain sebagai berikut.

- Himpunan kandidat (C) yang berisi kandidat yang akan dipilih pada setiap langkah.
- Himpunan solusi (S) yang berisi kandidat terpilih.
- Fungsi solusi (*solution function*) yang merupakan fungsi untuk mengecek apakah himpunan kandidat yang terpilih sudah memberikan solusi.
- Fungsi seleksi (*selection function*) yang memilih kandidat berdasarkan algoritma greedy tertentu.
- Fungsi kelayakan (*feasibility function*) yang memeriksa apakah kandidat yang terpilih dapat dimasukkan ke dalam himpunan solusi.
- Fungsi objektif (*objective function*) yang merupakan fungsi untuk memaksimumkan atau meminimumkan.

2.2. Alur Menjalankan Program

Untuk menjalankan permainan dan mengimplementasikan bot, diperlukan *starter pack* yang dapat diunduh melalui tautan di bawah ini.

Game engine starter pack:

<https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0>

Bot engine starter pack:

<https://github.com/haziqam/tubes1-IF2211-bot-starter-pack/releases/tag/v1.0.1>

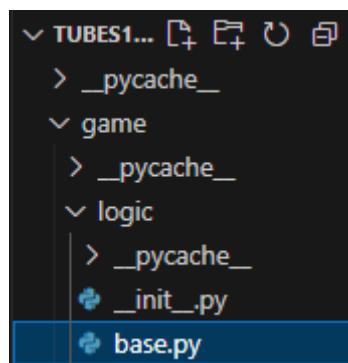
Untuk menjalankan program dapat mengikuti langkah-langkah pada tautan berikut:

Guidebook:

<https://docs.google.com/document/d/1L92Axb89yIkom0b24D350Z1QAr8rujvHof7-kXRAp7c/edit>

2.3. Alur pengembangan bot

Dari folder *tubes1-IF2211-bot-starter-pack-1.0.1*, masuk ke folder *game*, lalu masuk ke folder *game*, kemudian masuk ke folder *logic*. Pada folder *logic*, dapat dibuat sebuah file python baru yang akan meng-*inherit* class *BaseLogic* dari *base.py*.



Gambar 2.3.1. Lokasi base.py

Pengembangan dimulai dengan melakukan implementasi dari method *next_move(self, board_bot: GameObject, board: Board)*.

```
from abc import ABC
from typing import Tuple

from game.models import Board, GameObject

class BaseLogic(ABC):
    def next_move(self, board_bot: GameObject, board: Board) -> Tuple[int, int]:
        raise NotImplementedError()
```

Gambar 2.3.2. Class BaseLogic di base.py

```
def next_move(self, board_bot: GameObject, board: Board):
    start = time.time()
    if board_bot.properties.diamonds < 2:
        tackle = self.tackle(board_bot, board)
        if tackle[2]:
            return tackle[0], tackle[1]
    tempXY = self.getObjectsPosition(board_bot, board)
    delta_x = 0
    delta_y = 0
    print()
    threshold = 4
    count = 0
    self.targetX = tempXY[0]
    self.targetY = tempXY[1]
    startw = time.time()
    while(delta_x == 0 and delta_y == 0 and count < threshold):
        if tempXY[2]:
            print("TACKLE CUY")
```

Gambar 2.3.3. Modifikasi next_move() pada FirstToTheKeyFirstToTheEggBot.py yang telah meng-*inherit* class BaseLogic

BAB 3

Aplikasi Strategi Greedy

3.1. Alternatif Solusi

3.1.1. Greedy dengan prioritas red diamond

Greedy dengan prioritas *red diamond* terdekat adalah strategi yang mengutamakan pengambilan *red diamond* terdekat dari posisi saat itu. *Red diamond* yang menjadi target pada setiap *tick* akan dikalkulasi ulang, sehingga jika target *red diamond* awal ternyata sudah tidak ada, target *red diamond* terdekat akan berubah menyesuaikan dengan lokasi saat itu. Saat jumlah *diamond* pada *inventory* sudah lebih besar sama dengan 4, bot akan kembali ke base.

- **Mapping Elemen Greedy**

- Himpunan kandidat: semua *red diamond* yang ada pada *board*.
- Himpunan solusi: *red diamond* yang terpilih.
- Fungsi solusi: memeriksa apakah *red diamond* yang terpilih merupakan *red diamond* dengan posisi terdekat dari bot.
- Fungsi seleksi: memilih *red diamond* terdekat dari bot.
- Fungsi kelayakan: memeriksa apakah *inventory* masih cukup untuk mengambil *red diamond*.
- Fungsi objektif: jumlah *red diamond* maksimum.

- **Analisis Efisiensi Solusi**

Pada himpunan kandidat, dilakukan pengecekan semua *diamond* pada *board* untuk mencari *red diamond* dengan kompleksitas O(N).

- **Analisis Efektivitas Solusi**

Strategi ini mengutamakan pengumpulan *diamond* dengan poin yang lebih besar.

Strategi ini efektif apabila:

- *Red diamond* ada di dekat base atau bot;
- Perbandingan jumlah *red diamond* sebanding atau lebih banyak daripada *diamond* biasa; dan
- Kapasitas *inventory* berjumlah genap.

Strategi ini kurang efektif apabila:

- Lokasi *red diamond* yang tersisa jauh dari *base* atau *bot*;
- Perbandingan jumlah *red diamond* lebih sedikit dibandingkan dengan *diamond* biasa; dan
- Kapasitas *inventory* berjumlah ganjil sehingga tidak dapat mengambil *red diamond* dengan maksimal.

3.1.2. Greedy dengan perhitungan jarak *diamond* terdekat dari posisi bot

Greedy dengan prioritas *diamond* terdekat adalah strategi yang mengutamakan pengumpulan *diamond* terdekat dari posisi *bot* saat itu. *Diamond* yang menjadi target pada setiap *tick* akan dikalkulasi ulang, sehingga jika target *diamond* awal ternyata sudah tidak ada, maka target *diamond* terdekat akan berubah juga menyesuaikan dengan kondisi saat ini. Jika *diamond* yang dikumpulkan *bot* sudah mencapai lebih sama dengan 4, maka *bot* akan kembali ke *base*.

● **Mapping Elemen Greedy**

- Himpunan Kandidat: semua *diamond* yang ada di *board*.
- Himpunan Solusi: *diamond* yang terpilih.
- Fungsi Solusi: memeriksa apakah *diamond* yang terpilih merupakan *diamond* dengan posisi terdekat dari *bot*.
- Fungsi Seleksi: memilih *diamond* terdekat dengan *bot*.
- Fungsi Kelayakan: memeriksa apakah *diamond* pada *inventory* sudah lebih besar sama dengan 4.
- Fungsi Objektif: jumlah *diamond* maksimum.

● **Analisis Efisiensi Solusi**

Pada himpunan kandidat, dilakukan pengecekan semua *diamond* pada *board* untuk mencari *diamond* terdekat dengan kompleksitas O(N).

● **Analisis Efektivitas Solusi**

Strategi ini mengutamakan mengumpulkan *diamond* dengan posisi yang lebih dekat dari *bot*.

Strategi ini efektif apabila:

- *Diamond* berada di dekat *base* atau *bot*.

Strategi ini kurang efektif apabila:

- *Diamond* dengan posisi terdekat berada jauh dari *base*.

3.1.3. *Greedy* dengan perhitungan jarak *diamond* terdekat dari posisi *bot* dengan menggunakan *teleporter*

Algoritma *Greedy* ini akan mengumpulkan *diamond* yang jaraknya terdekat dari posisi *bot* terlebih dahulu. Untuk mengetahui jarak *diamond* yang terdekat, akan dilakukan perhitungan dengan 2 cara. Cara yang pertama adalah dengan menghitung jarak *diamond* ke *bot* jika *bot* berjalan dengan biasa. Cara yang kedua adalah menghitung jarak *diamond* ke *bot* jika memanfaatkan *teleporter*.

Semua *diamond* yang ada di *board* akan dilakukan perhitungan dengan kedua cara tersebut untuk menentukan *diamond* dengan jarak terdekat ke *bot*. Jika *diamond* yang terpilih adalah *diamond* dengan jarak terpendek jika menggunakan *teleporter*, maka tujuan *bot* adalah berjalan ke *teleporter* sehingga pada *tick* perhitungan berikutnya *diamond* tersebut akan menjadi *diamond* dengan jarak terpendek dengan cara jalan biasa.

● **Mapping Elemen Greedy**

- Himpunan Kandidat: semua *diamond* dan *teleporter* yang ada di *board*.
- Himpunan Solusi: *diamond* dan *teleporter* yang terpilih.
- Fungsi Solusi: memeriksa apakah *diamond* yang terpilih merupakan *diamond* dengan posisi terdekat dari *bot* dan memeriksa apakah *teleporter* yang terpilih menuju *diamond* yang terdekat dari *bot* jika menggunakan *teleporter*.
- Fungsi Seleksi: memilih *diamond* terdekat dengan *bot* atau *teleporter* yang akan membawa *bot* ke *diamond* terdekat.
- Fungsi Kelayakan: memeriksa apakah *diamond* pada *inventory* sudah lebih besar sama dengan 4.
- Fungsi Objektif: jumlah *diamond* maksimum.

● **Analisis Efisiensi Solusi**

Pada himpunan kandidat, dilakukan pengecekan semua *diamond* pada *board* dan dihitung jaraknya dengan *bot* menggunakan kedua cara untuk mencari *diamond* terdekat dengan kompleksitas O(N).

- **Analisis Efektivitas Solusi**

Strategi ini mengutamakan mengumpulkan *diamond* dengan posisi yang paling dekat dari *bot*.

Strategi ini efektif apabila:

- *Diamond* berada di dekat *bot* atau *base*.
- *Diamond* berada jauh dari *bot*, namun dekat jika menggunakan *teleporter*.

Strategi ini kurang efektif apabila:

- *Diamond* dan *teleporter* berada jauh dari *base*.

3.1.4. Greedy dengan perhitungan jarak *diamond* terdekat dari posisi *base*

Algoritma Greedy ini akan mengumpulkan diamond yang jaraknya terdekat dari posisi base. Diamond yang menjadi target pada setiap tick akan dikalkulasi ulang, sehingga jika target diamond awal ternyata sudah tidak ada atau muncul diamond baru yang jaraknya lebih dekat dengan base, maka diamond yang menjadi target akan berubah juga menyesuaikan dengan kondisi saat ini. Jika diamond yang dikumpulkan bot sudah mencapai lebih dari atau sama dengan 4, maka bot akan kembali ke base.

- **Mapping Elemen Greedy**

- Himpunan Kandidat: semua diamond yang ada di board.
- Himpunan Solusi: diamond yang terpilih.
- Fungsi Solusi: memeriksa apakah diamond yang terpilih merupakan diamond yang terdekat dengan posisi base.
- Fungsi Seleksi: memilih diamond yang terdekat dari base.
- Fungsi Kelayakan: memeriksa apakah diamond di inventory sudah lebih besar dari 3.
- Fungsi Objektif: jumlah diamond maksimum.

- **Analisis Efisiensi Solusi**

Pada himpunan kandidat, dilakukan pengecekan semua diamond pada board untuk mencari diamond terdekat dengan kompleksitas O(N).

- **Analisis Efektivitas Solusi**

Strategi ini mengutamakan mengumpulkan diamond dengan posisi terdekat dengan posisi base.

Strategi ini efektif apabila:

- Diamond berada di dekat base.

Strategi ini kurang efektif apabila:

- Diamond berada jauh dari base.

3.1.5. Greedy dengan tackle musuh

Greedy dengan *tackle* musuh adalah strategi mengumpulkan diamond dengan cara men-*tackle* musuh yang sudah mengumpulkan diamond dan mengambil diamond mereka. Posisi bot musuh akan dikalkulasikan dan posisi bot musuh terdekat dengan posisi bot kita dan mempunyai diamond akan menjadi target untuk di-*tackle*.

- **Mapping Elemen Greedy**

- Himpunan Kandidat: semua bot yang ada di board selain bot kita sendiri.
- Himpunan Solusi: bot yang terpilih untuk di tackle.
- Fungsi Solusi: memeriksa apakah bot tersebut mempunyai diamond.
- Fungsi Seleksi: memilih bot yang mempunyai diamond.
- Fungsi Kelayakan: memeriksa apakah diamond pada inventory kurang dari 5.
- Fungsi Objektif: mengumpulkan diamond secara maksimum.

- **Analisis Efisiensi Solusi**

Untuk mengecek posisi bot musuh, dilakukan pengecekan posisi semua bot musuh yang ada pada board dengan kompleksitas $O(N)$.

- **Analisis Efektivitas Solusi**

Algoritma Greedy ini merupakan strategi yang agresif.

Strategi ini efektif apabila:

- Jumlah bot musuh cukup banyak.
- Posisi diamond terdekat lebih jauh dari posisi bot musuh yang berada di sekitar bot kita.

Strategi ini kurang efektif apabila:

- Semua bot musuh menggunakan strategi mengumpulkan diamond dengan strategi *tackle* juga.
- Banyak diamond di sekitar bot kita.

3.1.6. Greedy dengan Prioritas Red Button

Algoritma Greedy dengan prioritas *red button* bertujuan untuk mengacak posisi *diamond* pada *board* dikarenakan sudah tidak ada *diamond* yang dekat dari bot dan cukup layak untuk diambil. Strategi ini digunakan ketika: jarak bot ke *red button* lebih dekat daripada jarak bot ke *diamond* terdekat; di sekitar *red button* tidak ada *diamond*; serta *inventory* bot tidak lebih dari tiga *diamond*.

- **Mapping Elemen Greedy**

- a. Himpunan kandidat: *red button* pada *board*.
- b. Himpunan solusi: *red button* yang jaraknya ke bot lebih dekat dibandingkan *diamond* terdekat ke bot.
- c. Fungsi Solusi: memeriksa apakah *red button* lebih dekat ke bot dibandingkan dengan jarak seluruh *diamond* ke bot.
- d. Fungsi Seleksi: memilih *red button* yang saat itu jaraknya relatif dekat dengan bot.
- e. Fungsi Kelayakan: memeriksa apakah ada *diamond* yang jaraknya lebih dekat dari bot.
- f. Fungsi Objektif: mendapatkan *diamond* maksimum.

- **Analisis Efisiensi Solusi**

Untuk membandingkan jarak dari semua *diamond* ke *bot* dan jarak dari *red button* ke *bot*, akan dilakukan pengecekan terhadap semua *diamond* dan *red button* yang ada dengan kompleksitas $O(N)$.

- **Analisis Efektivitas Solusi**

Strategi ini digunakan dengan harapan jika tidak ada *diamond* yang dekat dengan bot, namun ada *red button* yang jaraknya dekat dengan bot, bot akan mendatangi *red button*. Hal ini dilakukan dengan harapan saat *board* di reset akan ada *diamond* yang *spawn* di sekitar bot.

Strategi ini efektif apabila:

- Posisi *diamond* jauh dari posisi bot dan ada *red button* yang jaraknya lebih dekat dengan bot.

3.2. Strategi Terpilih

Berdasarkan analisis efektivitas dan efisiensi solusi dari alternatif solusi pada Bab 3.1, strategi yang terpilih adalah gabungan dari beberapa algoritma greedy yang telah dilakukan modifikasi agar dapat disatukan dengan baik. Algoritma greedy yang terpilih adalah greedy jarak diamond terdekat dengan posisi bot menggunakan teleporter, greedy dengan tackle musuh, greedy dengan *red button*. Fitur dan poin penting dari implementasi strategi greedy pada bot antara lain:

- Greedy dengan Tackle akan dilakukan jika ada bot lain yang bersebelahan dan diamond bot tersebut > 2 dan diamond kita saat ini < 2 .
- Penggunaan teleporter untuk efisiensi pergerakan saat akan mengambil diamond dan saat balik ke base.
- Greedy dengan memanfaatkan Red Button jika button adalah jarak terdekat dari posisi bot saat ini dan diamond pada inventory saat ini adalah < 4 dan maksimal 3 langkah dari Button tidak terdapat diamond.
- Mencari dan mendapatkan diamond terdekat dari posisi saat ini dan segera kembali ke base jika diamond sudah > 3 .
- Kembali ke base ketika selisih waktu sisa permainan dengan jarak yang diperlukan untuk kembali ke base sama dengan 2.

BAB 4

Implementasi dan Pengujian

4.1. Pseudocode

```
function getObjectsPosition(self,board_bot:GameObject,board:Board) → (integer,  
integer)
```

KAMUS LOKAL

```
object : GameObject  
  
step_in_tele1 : integer  
  
step_in_tele2 : integer  
  
temp_step : integer  
  
diamond_min_step_temp : integer  
  
Button_step_temp : integer  
  
button_step_tele1 : integer  
  
button_step_tele2 : integer  
  
step_to_center : integer  
  
min_step : integer  
  
target_X : integer  
  
target_Y : integer  
  
currX : integer  
  
currY : integer  
  
diamond_min_step : integer  
  
shortest_diamond_X : integer  
  
shortest_diamond_Y : integer  
  
n_diamond : integer  
  
button_step : integer
```

```

button_X : integer
button_Y : integer
button_radar : integer
diamond_in_button_radar : integer
isTele1 : Boolean

ALGORITMA
target_X ← 99
target_Y ← 99
currX ← board_bot.position.x
currY ← board_bot.position.y
diamond_min_step ← 99
shortest_diamond_X ← 99
shortest_diamond_Y ← 99
n_diamond ← 0
button_step ← 99
button_X ← 99
button_Y ← 99
button_radar ← 3
diamond_in_button_radar ← 0
isTele1 ← True
{ Iterasi objek }
for object in board.game_objects do
{ Dapatkan info diamond terdekat }
if object.type == "DiamondGameObject" then
    n_diamond ← n_diamond + 1
    if abs(object.position.x-button_X)+abs(object.position.y-button_Y) ≤
        button_radar then

```

```

diamond_in_button_radar ← diamond_in_button_radar + 1

{ Teleporter (objek bertipe teleporter selalu berada di awal sehingga
data teleporter sudah didapatkan terlebih dahulu) }

step_in_tele1 ← (abs(currX-self.tele1_X) + abs(currY-self.tele1_Y)) +
(abs(self.tele2_X-object.position.x) +
abs(self.tele2_Y-object.position.y))

step_in_tele2 ←
(abs(currX-self.tele2_X)+abs(currY-self.tele2_Y)+(abs(self.tele1_X-ob
ject.position.x)+abs(self.tele1_Y-object.position.y))

{ Original }

temp_step ← abs(object.position.y-board_bot.position.y) +
abs(object.position.x-board_bot.position.x)

{ Menentukan step terpendek }

diamond_min_step_temp ←
min(temp_step,step_in_tele1,step_in_tele2)

if diamond_min_step_temp < diamond_min_step then
    diamond_min_step ← diamond_min_step_temp

{ Penentuan posisi koordinat target }

if diamond_min_step == step_in_tele1 then
    shortest_diamond_X ← self.tele1_X
    shortest_diamond_Y ← self.tele1_Y
else if diamond_min_step == step_in_tele2 then
    shortest_diamond_X ← self.tele2_X
    shortest_diamond_Y ← self.tele2_Y
else
    shortest_diamond_X ← object.position.x
    shortest_diamond_Y ← object.position.y

{ Dapatkan info teleporter }

else_if object.type == "TeleportGameObject" then

```

```

{ objek bertipe teleporter yang pertama ditemukan saat loop dianggap
sebagai Tele1 }

if isTele1 then

    self.tele1_X ← object.position.x

    self.tele1_Y ← object.position.y

    self.tele1_step ←
        abs(object.position.x-currX)+abs(object.position.y-currY)

    isTele1 ← False

else

    self.tele2_X ← object.position.x

    self.tele2_Y ← object.position.y

    self.tele2_step ←
        abs(object.position.x-currX)+abs(object.position.y-currY)

{ Dapatkan info button }

else if object.type == "DiamondButtonGameObject":

    button_X ← object.position.x

    button_Y ← object.position.y

    button_step_temp ← abs(currX-button_X) + abs(currY-button_Y)

    button_step_tele1 ← (abs(currX-self.tele1_X) +
        abs(currY-self.tele1_Y)) + (abs(self.tele2_X-object.position.x) +
        abs(self.tele2_Y-object.position.y))

    button_step_tele2 ← (abs(currX-self.tele2_X) +
        abs(currY-self.tele2_Y)) + (abs(self.tele1_X-object.position.x) +
        abs(self.tele1_Y-object.position.y))

    button_step ←
        min(button_step_temp,button_step_tele2,button_step_tele1)

    if button_step == button_step_tele1 then

        button_X ← self.tele1_X

        button_Y ← self.tele1_Y

    else if button_step == button_step_tele2 then

```

```

button_X ← self.tele2_X

button_Y ← self.tele2_Y

else

    button_X ← object.position.x

    button_Y ← object.position.y

if n_diamond == 1 then

    { Jika sisa diamond pada board adalah 1 dan jumlah diamond saat ini adalah
    0, maka targetnya adalah pergi ke tengah dan bersiap untuk reset diamond
    setelah diamond terakhir diambil orang lain atau pergi ke button jika button
    lebih dekat dengan kita }

    if board_bot.properties.diamonds == 0 then

        step_to_center ← abs(7-currX)+abs(7-currY)

        if step_to_center < button_step then

            target_X ← 7

            target_Y ← 7

        else

            target_X ← button_X

            target_Y ← button_Y

    { Jika sisa diamond pada board adalah 1 dan jumlah diamond saat ini
    > 0, maka akan lebih baik mengamankan terlebih dahulu ke base }

    else

        target_X ← board_bot.properties.base.x

        target_Y ← board_bot.properties.base.y

    else

        min_step ← min(diamond_min_step,button_step)

    { Pergi ke button jika dan hanya jika:

        1. Button adalah yang terdekat dan

        2. Tidak ada diamond sama sekali di sekitar button dengan jarak 3 langkah
        dari button dan

```

3. Diamond saat ini < 4 }

```

if min_step == button_step and diamond_in_button_radar == 0 and
board_bot.properties.diamonds < 4 then

    target_X ← button_X

    target_Y ← button_Y

else

    target_X ← shortest_diamond_X

    target_Y ← shortest_diamond_Y

→ (target_X, target_Y)

```

function tackle(self,board_bot:GameObject,board:Board) → (integer, integer, boolean)

{ Fungsi untuk mengiterasi objek bot lain dan menentukan apakah akan mencoba melakukan tackle atau tidak }

KAMUS LOKAL

```

board_bot : GameObject

board : Board

bot : GameObject

currX : integer

currY : integer

isTackle : boolean

dx : integer

dy : integer

```

ALGORITMA

```

currX ← board_bot.position.x

currY ← board_bot.position.y

isTackle ← False

dx ← 0

```

```

dy ← 0

{ Iterasi bot }

for bot in board.bots do

    { Hanya bot dengan diamond > 2 yang diperhitungkan }

    if bot.properties.diamonds > 2 then

        tackleX ← bot.position.x-currX

        tackleY ← bot.position.y-currY

        if (tackleX == 1 or tackleX == -1) and tackleY==0 then

            dx ← tackleX

            isTackle ← True

            break

        if (tackleY == 1 or tackleY == -1) and tackleX==0 then

            dy ← tackleY

            isTackle ← True

            break

    → (dx, dy, isTackle)

```

function next_move(self, board_bot then GameObject, board then Board) → (integer, integer)

KAMUS LOKAL

```

board_bot : GameObject
board : Board
tackle : list
tempXY : tuple
delta_x : integer
delta_y : integer
threshold : integer
count : integer
self.targetX : integer
self.targetY : integer

```

currX : integer
 currY : integer
 original_step : integer
 tele1_step : integer
 tele2_step : integer
 min_target_step : integer

ALGORITMA

```

{ Mencoba tackle hanya jika diamond saat ini < 2 }
if board_bot.properties.diamonds < 2 then
    tackle ← self.tackle(board_bot,board)
    if tackle[2] then
        → (tackle[0], tackle[1])

{ Iterate objects }
tempXY ← self.getObjectsPosition(board_bot,board)
delta_x ← 0
delta_y ← 0
threshold ← 4 { Maximum iteration to decide delta_x and delta_y }
count ← 0
self.targetX ← tempXY[0]
self.targetY ← tempXY[1]
{ Iterate }
while delta_x == 0 and delta_y == 0 and count < threshold) do
    currX ← board_bot.position.x
    currY ← board_bot.position.y
    { Hitung langkah kembali ke base dengan cara biasa dan dengan
    teleporter }
    original_step ← abs(board_bot.position.x -
    board_bot.properties.base.x) + abs(board_bot.position.y -
    board_bot.properties.base.y)
    tele1_step ← (abs(currX - self.tele1_X) + abs(currY - self.tele1_Y)) +
    (abs(board_bot.properties.base.x - self.tele2_X) +
    abs(board_bot.properties.base.y - self.tele2_Y))
    tele2_step ← (abs(currX - self.tele2_X) + abs(currY - self.tele2_Y)) +
    (abs(board_bot.properties.base.x - self.tele1_X) +
    abs(board_bot.properties.base.y - self.tele1_Y))
    min_target_step ← min(original_step,tele1_step,tele2_step)
    { Endgame : Jika selisih antara sisa waktu permainan }

```

```

{ dan langkah untuk kembali ke base ← 2, maka kembali ke base }
if (((board_bot.properties.milliseconds_left//1000)) - min_target_step)
≤ 2 then
    if min_target_step == tele1_step then
        self.targetX ← self.tele1_X
        self.targetY ← self.tele1_Y
    else if min_target_step == tele2_step then
        self.targetX ← self.tele2_X
        self.targetY ← self.tele2_Y
    else
        self.targetX ← board_bot.properties.base.x
        self.targetY ← board_bot.properties.base.y
    if self.targetX ≠ 99 then
        if self.targetX < currX then
            delta_x ← -1
            break
        else if self.targetX > currX then
            delta_x ← 1
            break
        else
            self.targetX ← 99
    if self.targetY ≠ 99 then
        if self.targetY < currY then
            delta_y ← -1
            break
        else if self.targetY > currY then
            delta_y ← 1
            break
        else
            self.targetY ← 99
{ Jika diamond di inventory sudah ada 4 atau 5 maka kembali ke base
}
else if board_bot.properties.diamonds > 3 then
{ Hitung langkah kembali ke base dengan cara biasa dan dengan
teleporter }
original_step ← abs(currX - board_bot.properties.base.x) + abs(currY
- board_bot.properties.base.y)

```

```

tele1_step ← (abs(currX - self.tele1_X) + abs(currY - self.tele1_Y)) +
(abs(board_bot.properties.base.x - self.tele2_X) +
abs(board_bot.properties.base.y - self.tele2_Y))
tele2_step ← (abs(currX - self.tele2_X) + abs(currY - self.tele2_Y)) +
(abs(board_bot.properties.base.x - self.tele1_X) +
abs(board_bot.properties.base.y - self.tele1_Y))
min_target_step ← min(original_step,tele1_step,tele2_step)
if min_target_step == tele1_step then
    self.targetX ← self.tele1_X
    self.targetY ← self.tele1_Y
else if min_target_step == tele2_step then
    self.targetX ← self.tele2_X
    self.targetY ← self.tele2_Y
else
    self.targetX ← board_bot.properties.base.x
    self.targetY ← board_bot.properties.base.y
if self.targetX ≠ 99 then
    if self.targetX < currX then
        delta_x ← -1
        break
    else if self.targetX > currX then
        delta_x ← 1
        break
    else
        self.targetX ← 99
if self.targetY ≠ 99 then
    if self.targetY < currY then
        delta_y ← -1
        break
    else if self.targetY > currY then
        delta_y ← 1
        break
    else
        self.targetY ← 99
else
    if self.targetX ≠ 99 then
        if self.targetX < currX then

```

```

        delta_x ← -1
        break
    else if self.targetX > currX then
        delta_x ← 1
        break
    else
        self.targetX ← 99
    if self.targetY ≠ 99 then
        if self.targetY < currY then
            delta_y ← -1
            break
        else if self.targetY > currY then
            delta_y ← 1
            break
        else
            self.targetY ← 99
    if self.targetX ≠ 99 then
        if self.targetX < currX then
            delta_x ← -1
            break
        else if self.targetX > currX then
            delta_x ← 1
            break
        else
            self.targetX ← 99
    if self.targetY ≠ 99 then
        if self.targetY < currY then
            delta_y ← -1
            break
        else if self.targetY > currY then
            delta_y ← 1
            break
        else
            self.targetY ← 99
    count + ← 1
{ Handling jika ternyata setelah hasil iterasi delta_x ← delta_y }
if delta_x == delta_y then

```

```

if currX == 0 then
    delta_x ← 1
    delta_y ← 0
else if currX == 14 then
    delta_x ← -1
    delta_y ← 0
else if currY == 0 then
    delta_x ← 0
    delta_y ← 1
else if currY == 14 then
    delta_x ← 0
    delta_y ← -1
else
    delta_x ← 1
    delta_y ← 0
→ (delta_x, delta_y)

```

4.2. Pengujian

Metode pengujian dilakukan sebanyak 10 kali dan pada setiap pertandingan, map akan di reset (tidak melanjutkan sisa map dari pertandingan terakhir).

Pengujian dilakukan dengan 4 bot:

- Bot stima (Strategi yang terpilih)

Fitur:

- Tackle (jika ada bot lain yang bersebelahan dan diamond bot tersebut > 2 dan diamond kita saat ini < 2).
- Penggunaan teleporter untuk efisiensi pergerakan.
- Memanfaatkan Button (jika memang button adalah jarak terdekat dari posisi saat ini dan diamond pada inventory saat ini adalah < 4 dan maksimal 3 langkah dari Button tidak terdapat diamond).
- Mendapatkan diamond terdekat dari posisi saat ini dan segera kembali ke base jika diamond sudah > 3 .
- Kembali ke base ketika selisih waktu sisa permainan dengan jarak yang diperlukan untuk kembali ke base sama dengan 2.

- Bot stima1 :

Fitur:

- Tackle (jika ada bot lain yang bersebelahan dan diamond bot tersebut > 2 dan diamond kita saat ini < 2).
- Penggunaan teleporter untuk efisiensi pergerakan.
- Mendapatkan diamond terdekat dari posisi saat ini dan segera kembali ke base jika diamond sudah > 3 .
- Kembali ke base ketika selisih waktu sisa permainan dengan jarak yang diperlukan untuk kembali ke base sama dengan 2.

- Bot stima2 :

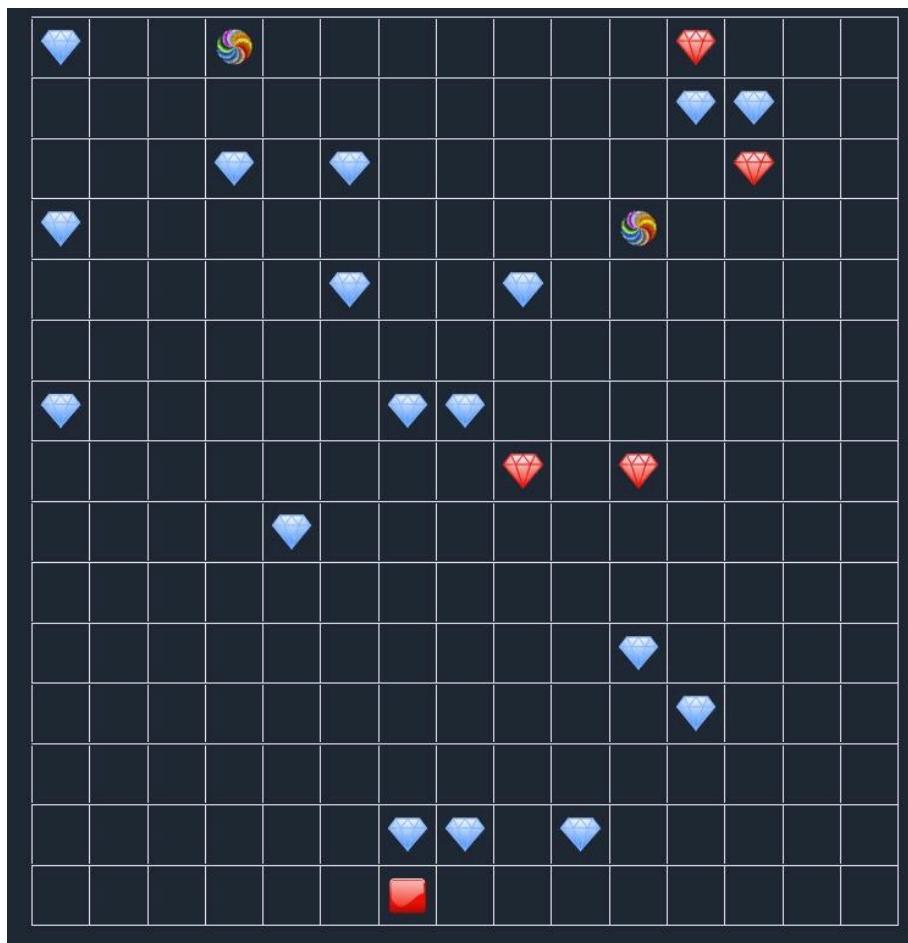
Fitur: Sama seperti bot stima, tetapi tidak ada fitur tackle.

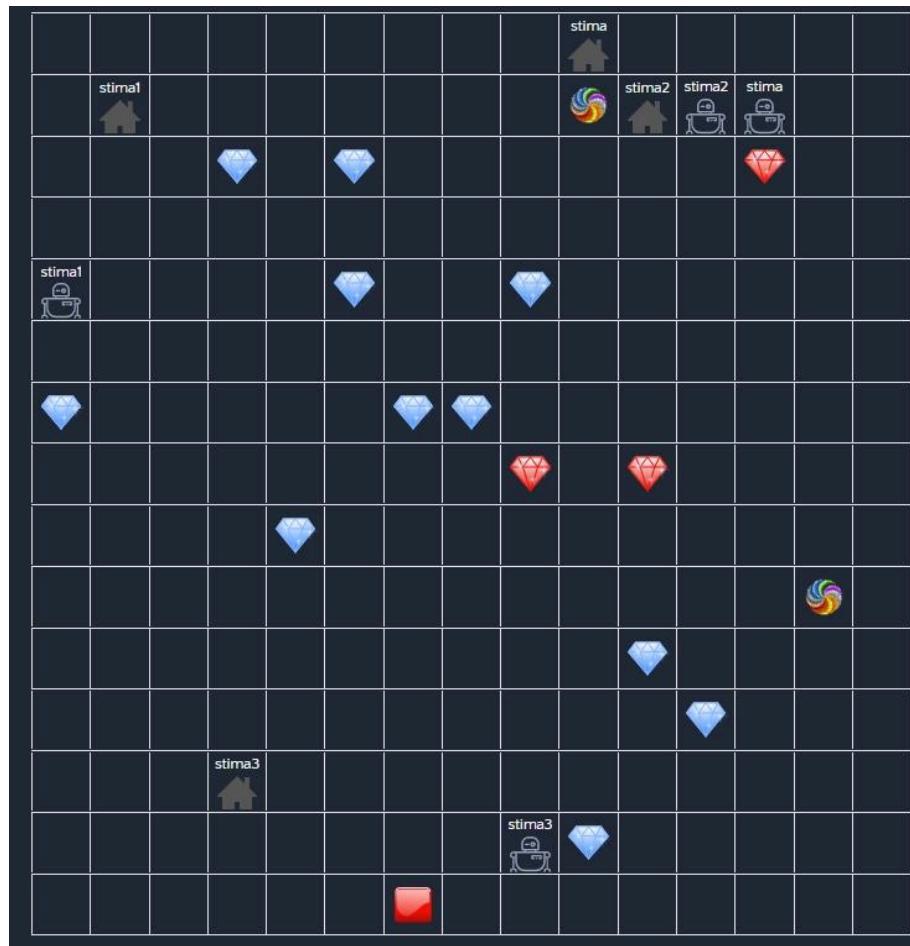
- Bot stima3 :

Fitur:

- Mendapatkan diamond terdekat dari posisi saat ini dan segera kembali ke base jika diamond sudah > 3 .
- Kembali ke base ketika selisih waktu sisa permainan dengan jarak yang diperlukan untuk kembali ke base sama dengan 2.

Pertandingan Ke-1

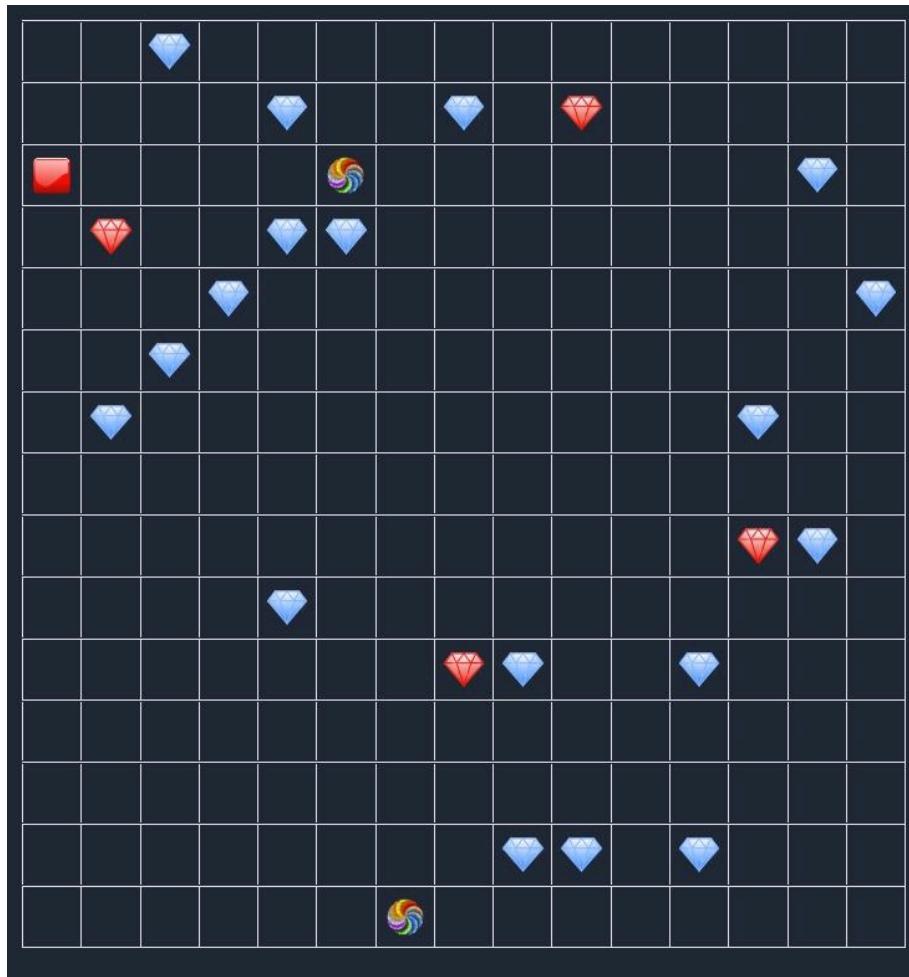


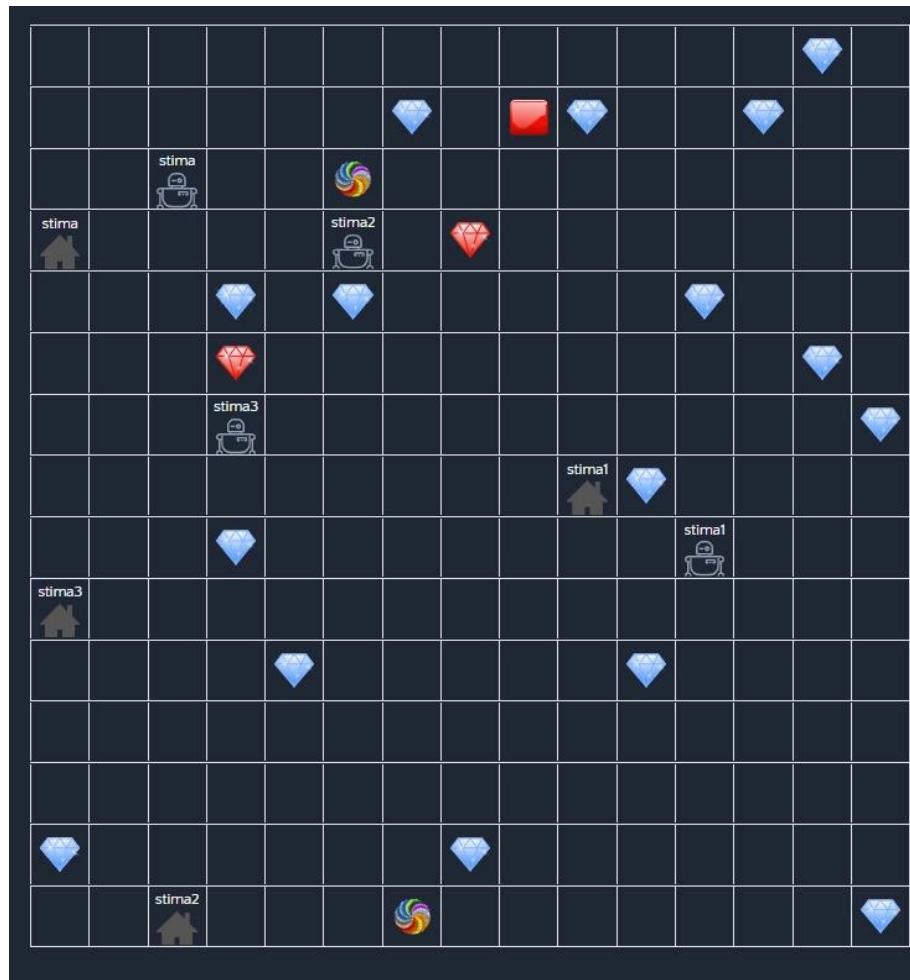


Final Score

Name	Score
stima1	13
stima	9
stima3	8
stima2	7

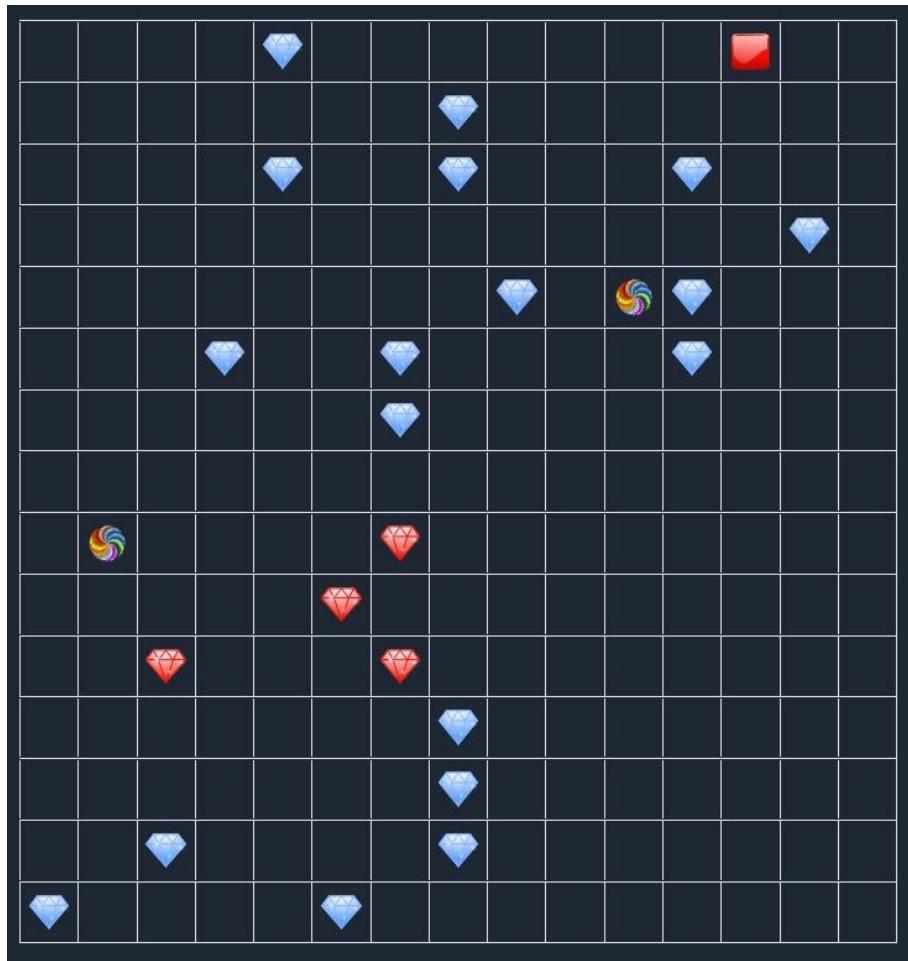
Pertandingan Ke-2

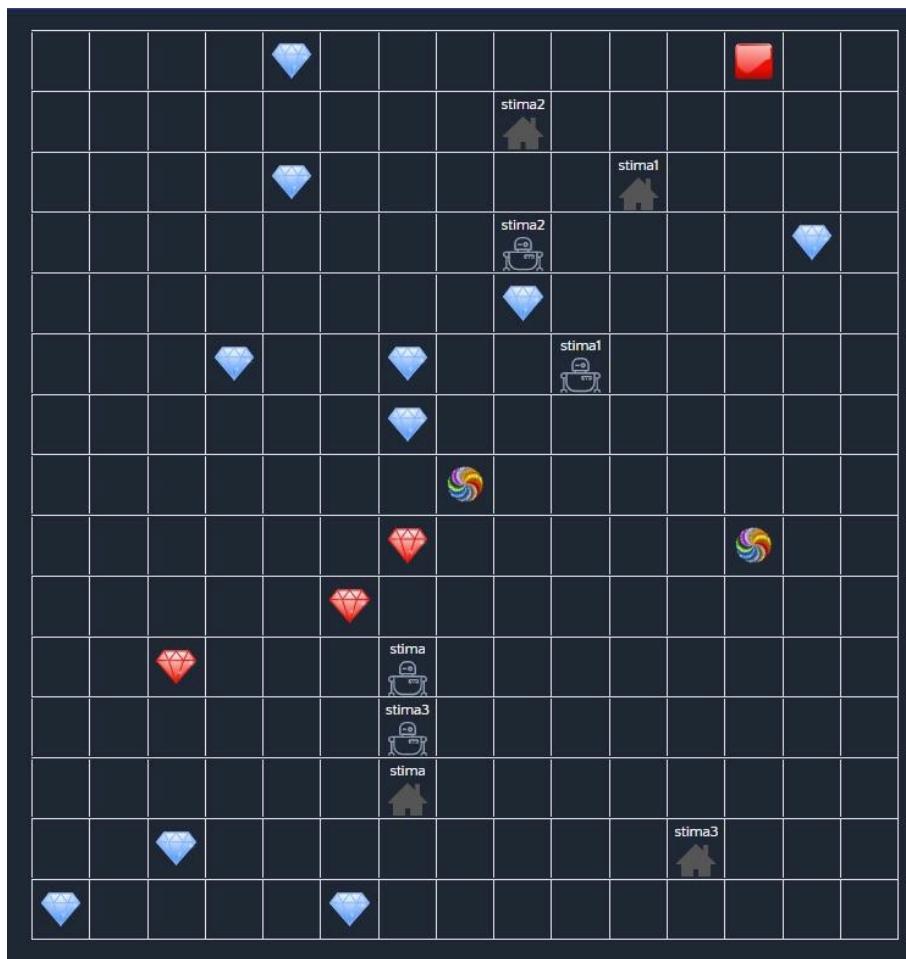




Final Score	
Name	Score
stima3	11
stima2	8
stima	7
stima1	1

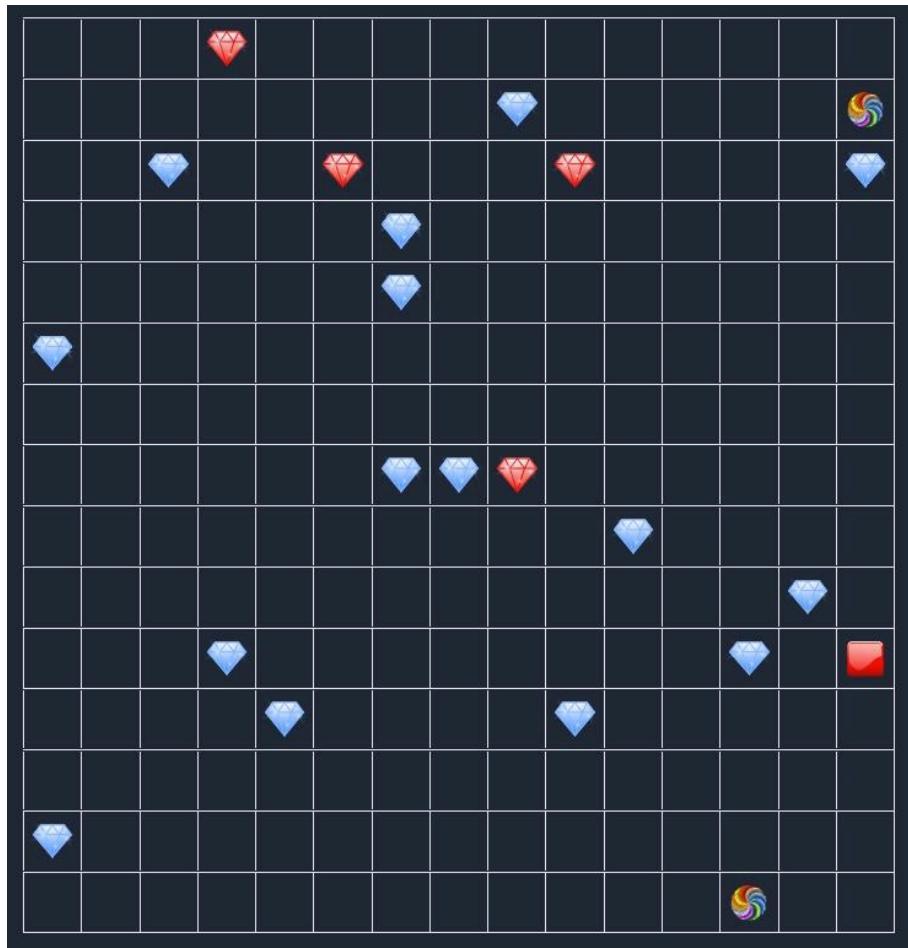
Pertandingan Ke-3

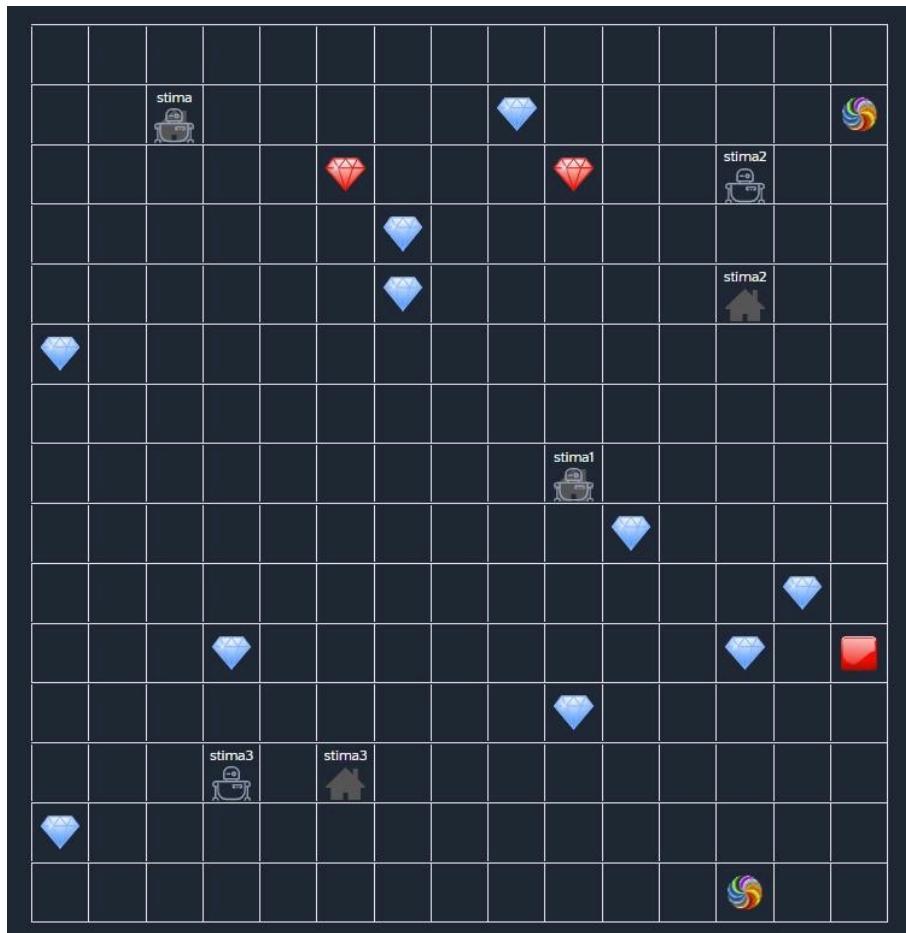


**Final Score**

Name	Score
stima2	16
stima	14
stima3	9
stima1	7

Pertandingan Ke-4

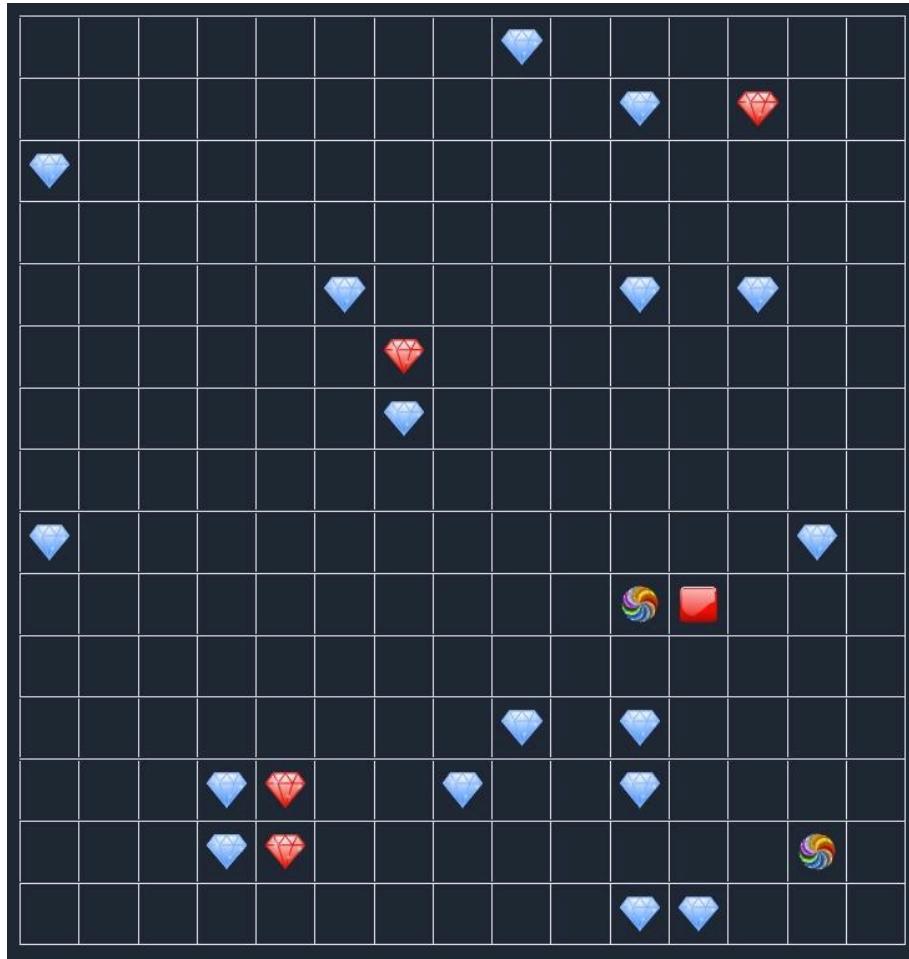


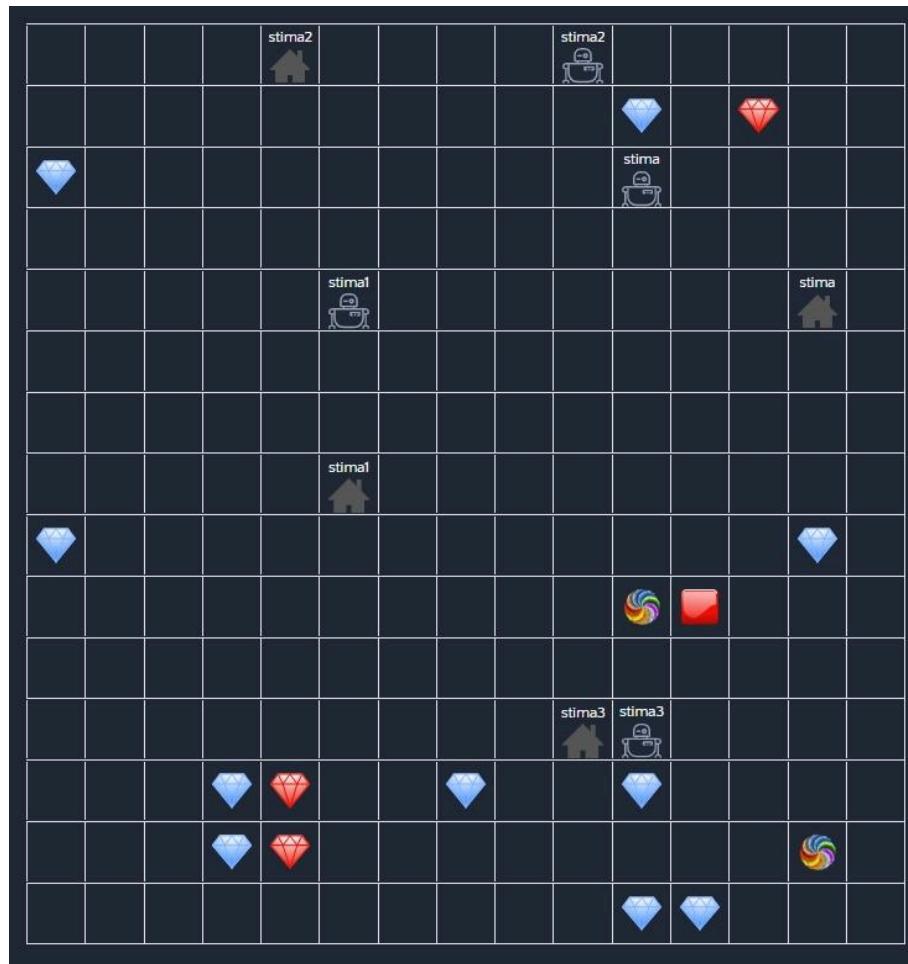


Final Score

Name	Score
stima	18
stima3	11
stima2	8
stima1	8

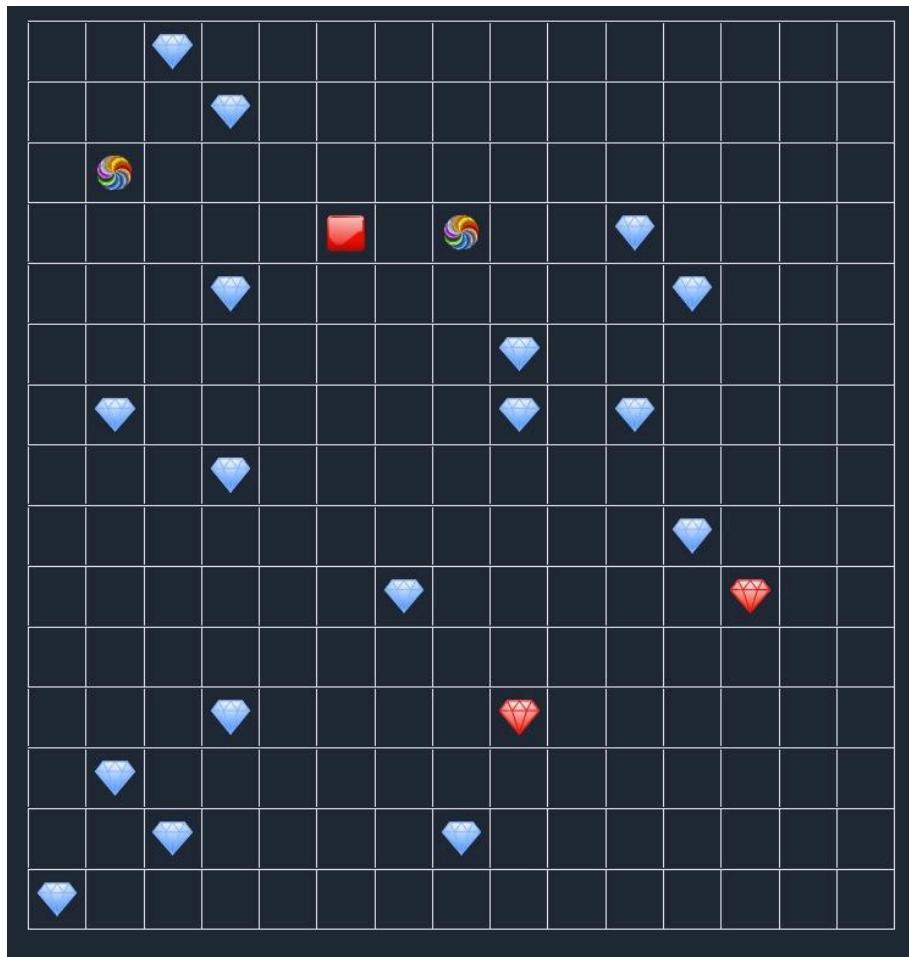
Pertandingan Ke-5

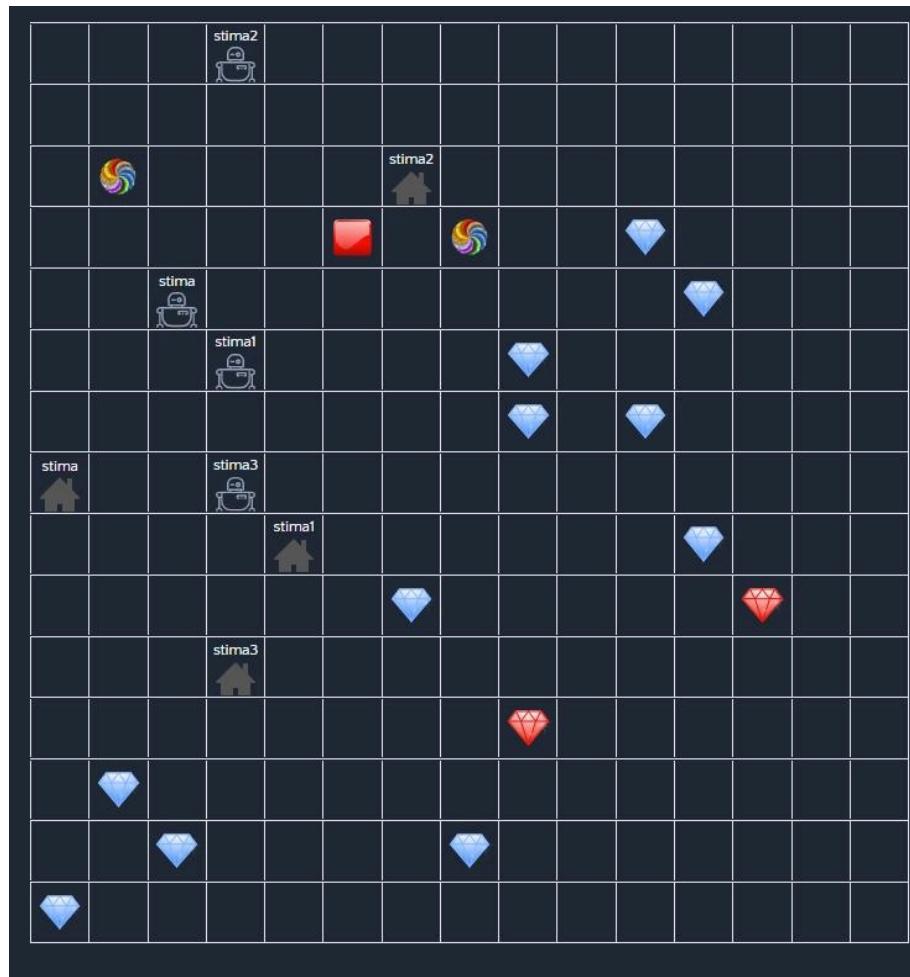




Final Score	
Name	Score
stima	18
stima3	16
stima1	12
stima2	8

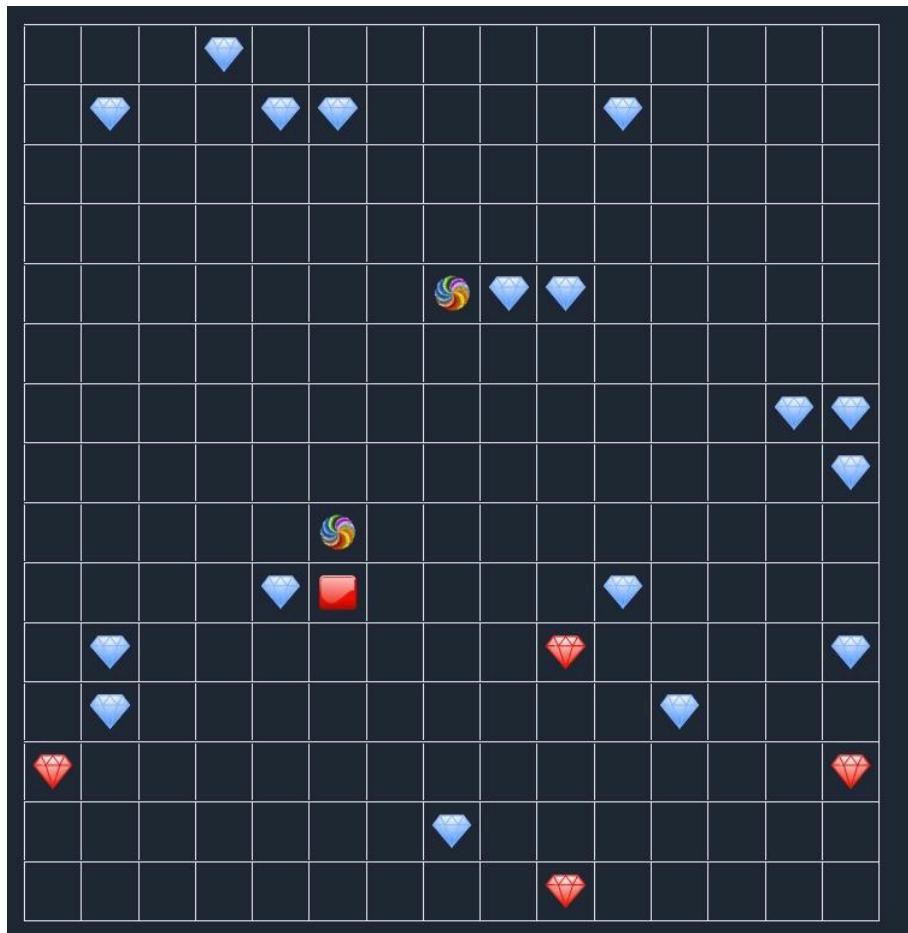
Pertandingan Ke-6

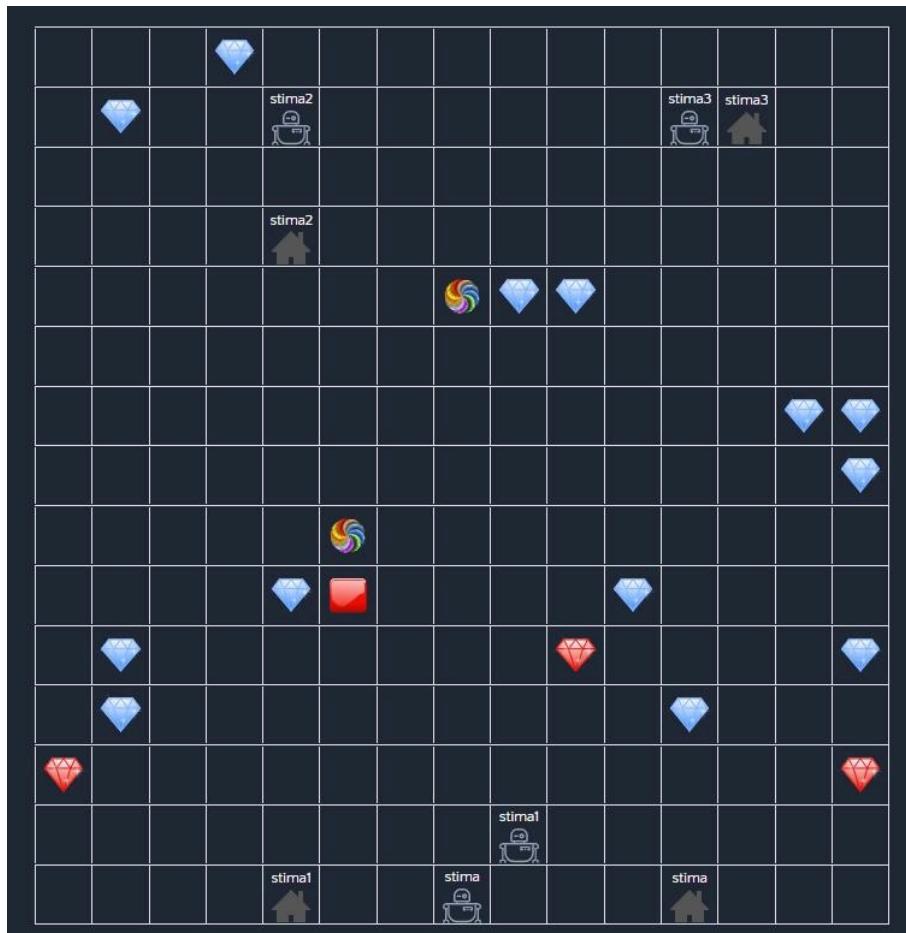


**Final Score**

Name	Score
stima3	12
stima2	10
stima	9
stima1	2

Pertandingan Ke-7

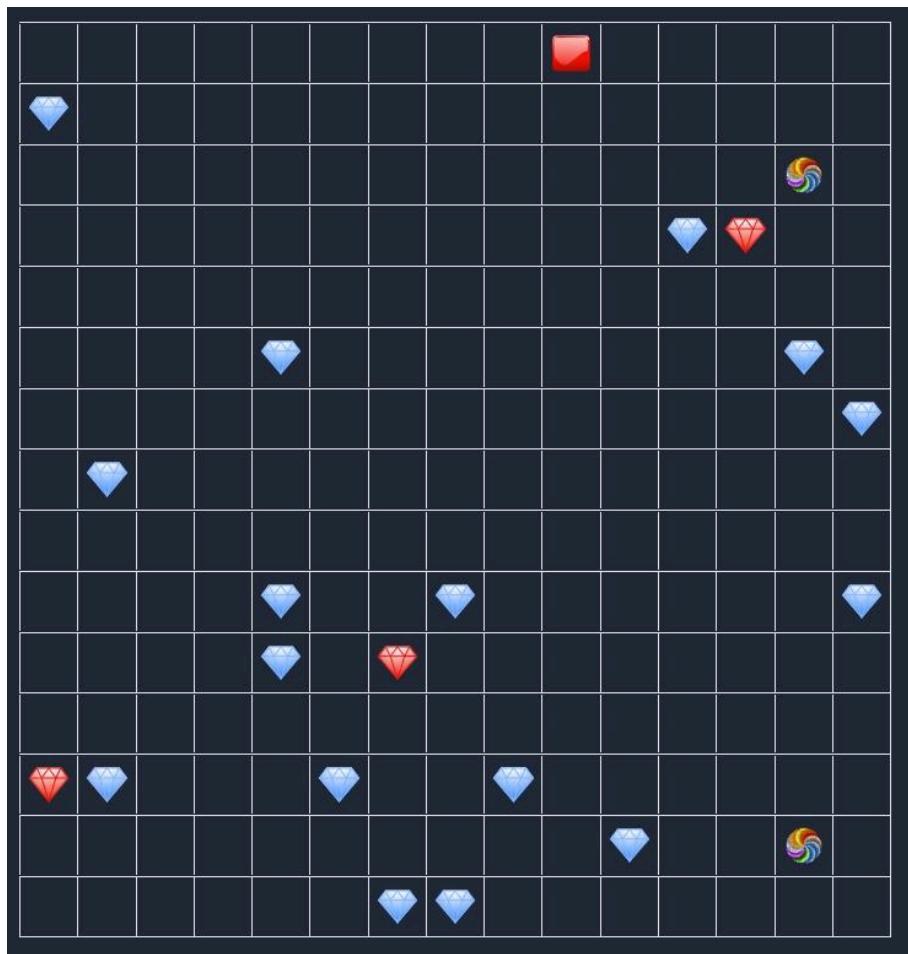


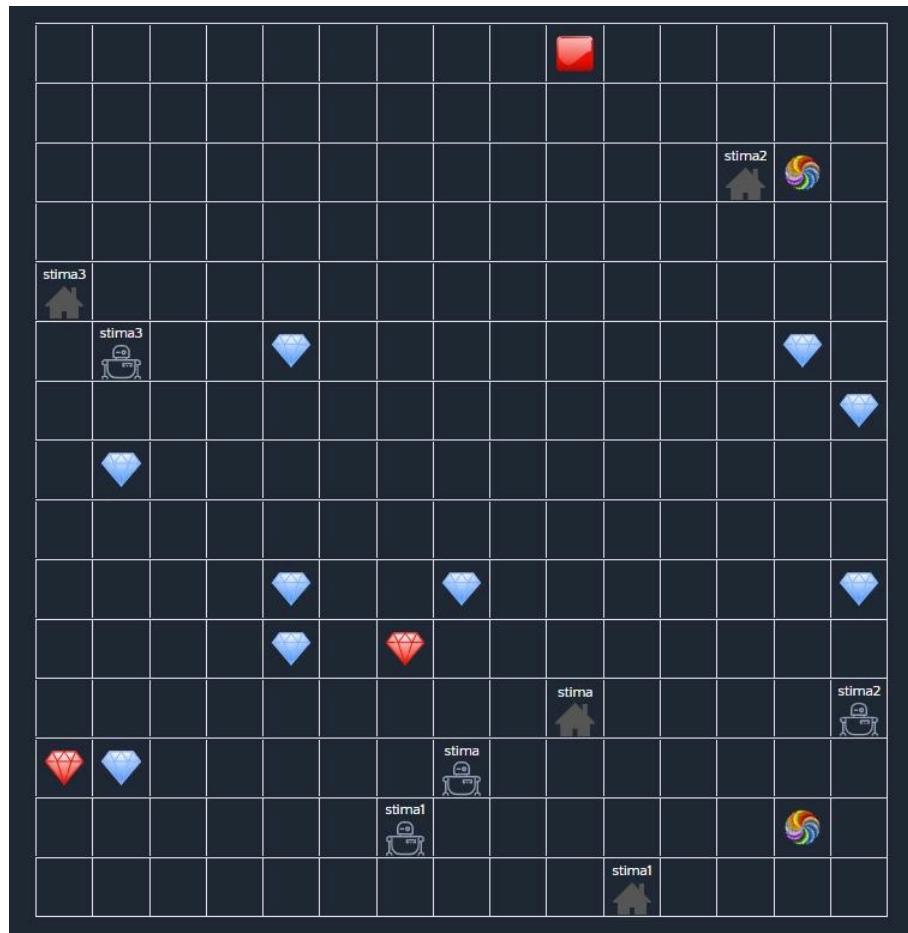


Final Score

Name	Score
stima2	13
stima	11
stima3	7
stima1	5

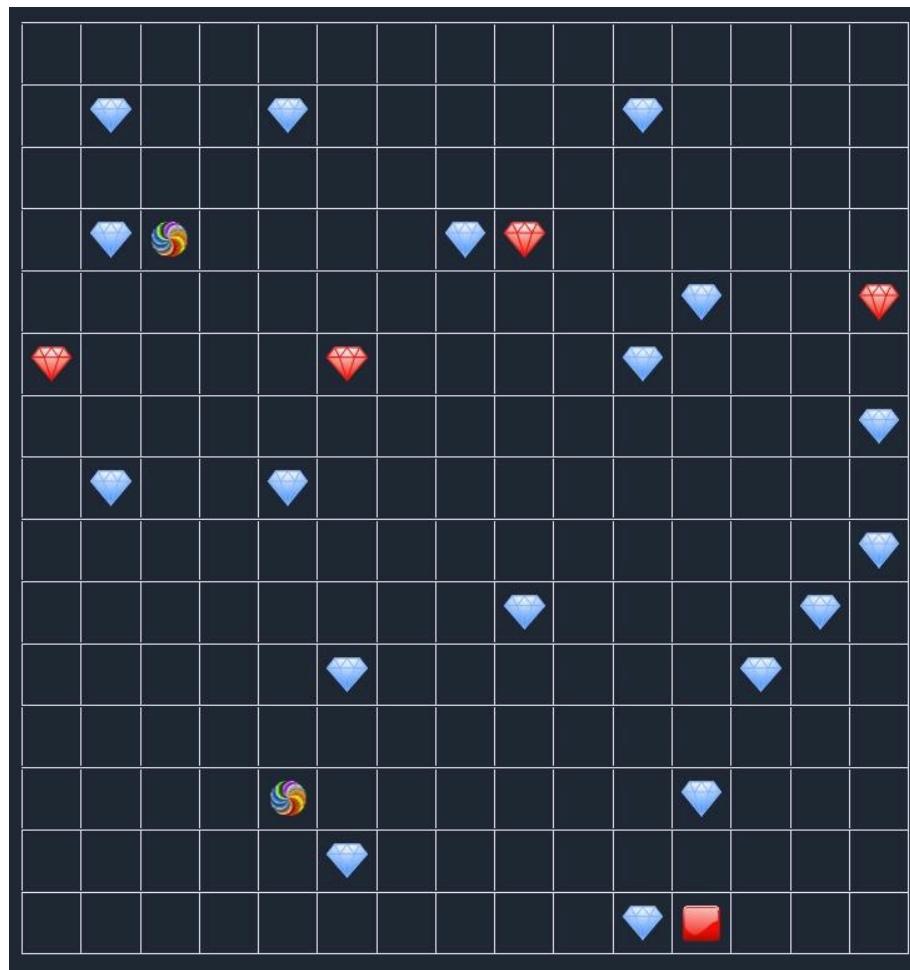
Pertandingan Ke-8

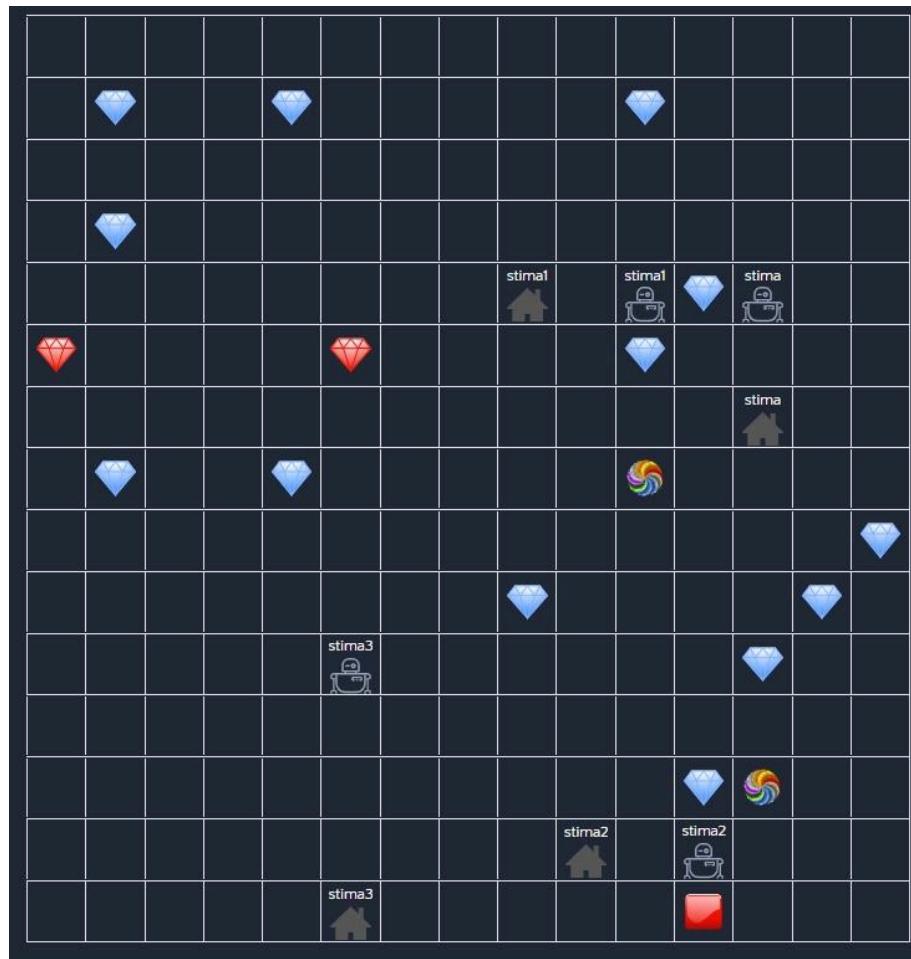


**Final Score**

Name	Score
stima	15
stima2	14
stima3	8
stima1	8

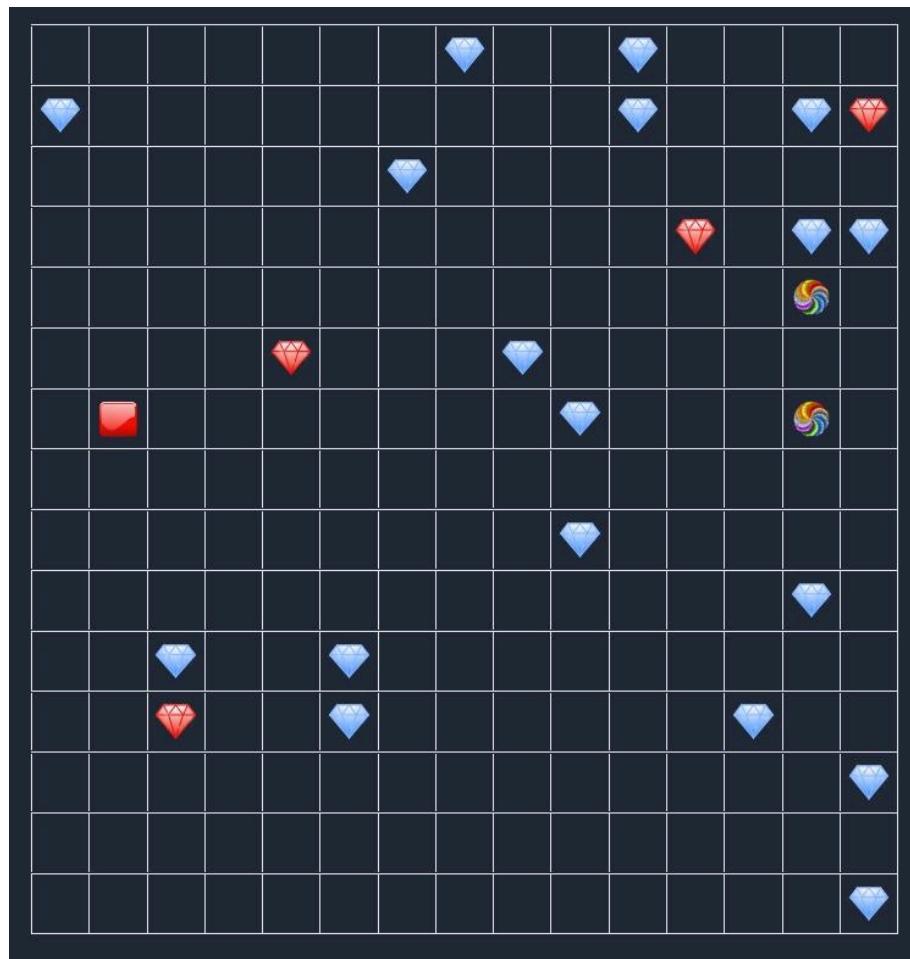
Pertandingan Ke-9

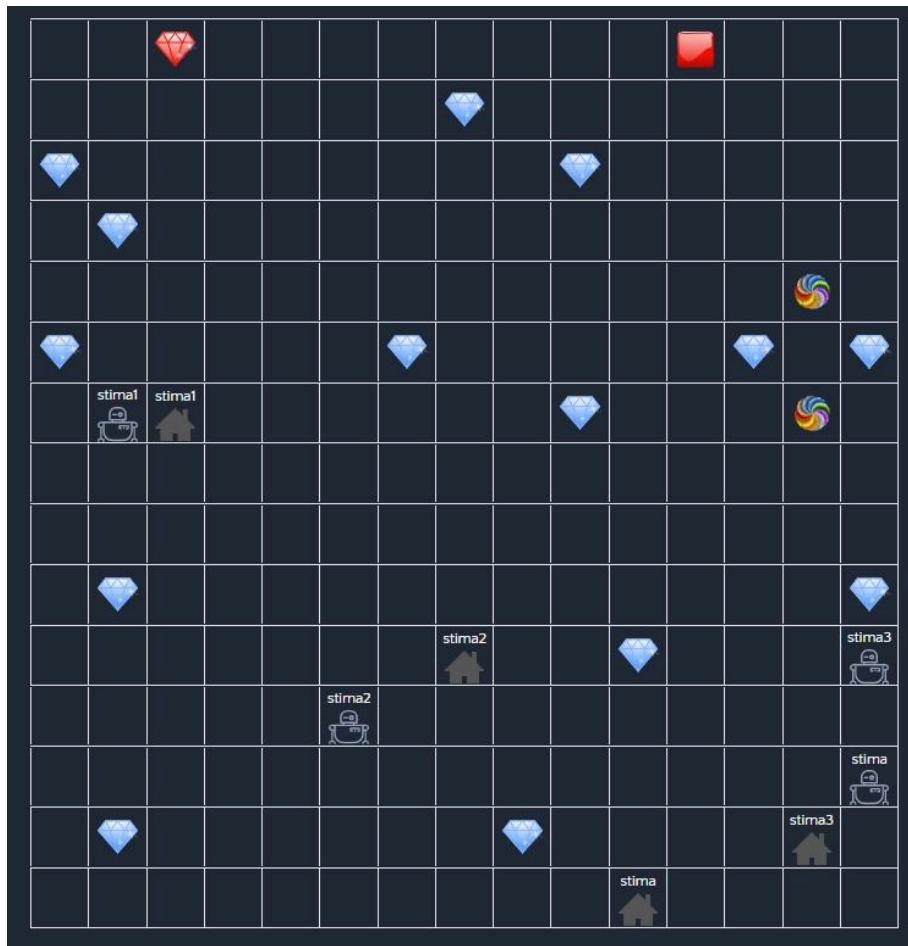




Final Score	
Name	Score
stima1	15
stima	9
stima3	7
stima2	2

Pertandingan Ke-10



**Final Score**

Name	Score
stima2	10
stima3	10
stima	9
stima1	4

4.3. Hasil dan Pembahasan

Hasil dari 10 Pertandingan												
Nama Bot	Jumlah Kemenangan	Score Pertandingan Ke-										Rata-Rata Score
		1	2	3	4	5	6	7	8	9	10	
stima	3	9	7	14	18	18	9	11	15	9	9	11,9
stima1	2	13	1	7	8	12	2	5	14	15	4	8,1
stima2	2,5	7	8	16	8	8	10	13	8	2	10	9,0
stima3	2,5	8	11	9	11	16	12	7	8	7	10	9,9

Pada tabel terdapat bot yang mendapatkan jumlah kemenangan 2,5, yaitu stima2 dan stima3. Hal tersebut karena 0,5 didapatkan dari pertandingan ke-10 dimana score dari stima2 dan stima3 adalah sama. Bot stima adalah bot yang paling unggul berdasarkan jumlah kemenangan, yaitu 3 kemenangan. Jika dilihat berdasarkan rata-rata score, bot stima juga merupakan bot yang paling unggul dengan rata-rata score terbesar yaitu 11,9. Berdasarkan hasil tersebut dapat disimpulkan bahwa bot stima adalah bot yang paling unggul.

BAB 5

Kesimpulan dan Saran

5.1. Kesimpulan

Dalam tugas besar IF2211 Strategi Algoritma ini, kami berhasil mengembangkan sebuah bot untuk permainan Diamonds dengan menggunakan algoritma Greedy. Strategi yang digunakan merupakan gabungan dari beberapa strategi yang memiliki prioritas sesuai dengan kondisi tertentu.

Strategi greedy yang terpilih dapat dibilang berhasil dalam membuat bot yang cukup baik pada permainan Diamonds. Gabungan dari beberapa strategi greedy ini berhasil untuk mencari solusi yang optimal lokal dalam berbagai macam kondisi. Namun, terkadang algoritma greedy gagal mendapatkan solusi optimum global sehingga mendapatkan score yang kecil.

Permainan Diamonds juga memiliki unsur keberuntungan yang mempengaruhi hasil permainan, namun penerapan algoritma Greedy yang baik dapat mengurangi dampak dari unsur keberuntungan tersebut dan menghasilkan hasil yang lebih konsisten. Selain algoritma greedy, algoritma lain yang dapat digunakan adalah algoritma *exhaustive search* untuk mengecek setiap kemungkinan *path* yang dapat diambil oleh bot untuk mendapatkan solusi optimum global.

5.2. Saran

Saran untuk penggerjaan tugas besar ini adalah:

- Penggerjaan bot lebih komunikatif lagi agar penggerjaannya dapat lebih cepat selesai.
- Penggerjaan tidak terlalu dekat dengan deadline agar hasil bisa lebih baik lagi
- Algoritma pada bot dapat dikembangkan lebih baik lagi dari segi optimasi dan fitur-fitur yang ada.

Lampiran

Link Repository:

https://github.com/evelynnn04/Tubes1_FirstToTheKeyFirstToTheEgg.git

Link Video:

<https://youtu.be/zI37sqoMJhw?si=2vp8B4Ub8gFk4WLA>

Daftar Pustaka

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf)