Tugas Besar I IF2211 Strategi Algoritma

Semester II Tahun 2020/2021

Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan “Worms”

Oleh:

13519070 - Mhd. Hiro Agayeff Muslion

13519071 - Farhan Nur Hidayat Denira

13519072 - Hanif Arroisi Mukhlis

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

Pada tugas besar kali ini, anda diminta untuk membuat sebuah *bot* untuk bermain permainan Worms yang telah dijelaskan sebelumnya. Untuk memulai, anda dapat mengikuti panduan singkat sebagai berikut.

1. *Download latest release starter pack.zip* dari tautan berikut. https://github.com/EntelectChallenge/2019-Worms/releases/tag/2019.3.2.
2. Untuk menjalankan permainan, kalian butuh beberapa *requirement* dasar sebagai berikut.
    A. Java (minimal Java 8): https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html
    B. IntelIJ IDEA: https://www.jetbrains.com/idea/
    C. NodeJS: https://nodejs.org/en/download/
3. Untuk menjalankan permainan, kalian dapat membuka file “run.bat” (Untuk *Windows/Mac* dapat buka dengan *double-click*, Untuk Linux dapat menjalankan *command* “make run”).
4. Secara *default*, permainan akan dilakukan diantara *reference bot* (*default*-nya berbahasa *JavaScript*) dan starter bot yang disediakan. Untuk mengubah hal tersebut, silahkan edit file “game-runner-config.json”. Anda juga dapat mengubah file “bot.json” untuk mengatur informasi terkait bot anda.
5. Silahkan bersenang-senang dengan memodifikasi bot yang disediakan di *starter-bot*. Ingat bahwa bot kalian harus menggunakan bahasa **Java** dan di-*build* menggunakan **IntelIJ**. **Dilarang** menggunakan kode program tersebut untuk pemainnya atau kode program lain yang diunduh dari Internet. Mahasiswa harus membuat program sendiri, tetapi belajar dari program yang sudah ada tidak dilarang.
6. (Optional) Anda dapat melihat hasil permainan dengan menggunakan *visualizer* berikut https://github.com/dlweatherhead/entelect-challenge-2019-visualiser/releases/tag/v1.01
7. Untuk referensi lebih lanjut, silahkan eksplorasi di tautan berikut.

Strategi *greedy* yang diimplementasikan tiap kelompok harus dikaitkan dengan fungsi objektif dari permainan itu sendiri, yaitu memenangkan permainan dengan cara mengeliminasi seluruh worms lawan dengan senjata dan skill yang sudah disediakan dalam permainan. Salah satu contoh pendekatan *greedy* yang bisa digunakan (pendekatan tak terbatas pada contoh ini saja) adalah menyerang pasukan lawan dengan senjata dengan *hitpoint* / *damage* terbesar. Buatlah strategi *greedy* terbaik, karena setiap “pemain” dari masing-masing kelompok akan diadu satu sama lain dalam suatu kompetisi Tubes 1 (secara daring).

Strategi *greedy* harus dituliskan secara eksplisit pada laporan, karena akan diperiksa pada saat demo apakah strategi yang dituliskan sesuai dengan yang diimplementasikan. Tiap kelompok dapat menggunakan kreativitas mereka dalam menyusun strategi greedy untuk memenangkan permainan. Implementasi pemain harus dapat dijalankan pada game engine yang telah disebutkan pada spesifikasi tugas besar, serta dapat dikompetisikan dengan pemain dari kelompok lain.

BAB II
LANDASAN TEORI

Algoritma greedy merupakan metode yang paling populer untuk memecahkan persoalan optimasi. Greedy berarti tamak, rakus, atau loba. Sesuai dengan artinya, prinsip dari algoritma greedy adalah memilih keputusan terbaik pada setiap state atau langkah (akan menghasilkan keputusan optimum lokal) dengan harapan akan menghasilkan keputusan optimum global. Namun, apabila keputusan terbaik pada suatu langkah ternyata tidak mengarahkan pada optimum global, tidak dapat dilakukan peninjauan kembali (backtracking) terhadap keputusan tersebut.

Algoritma greedy memiliki beberapa elemen yang harus dipenuhi, yaitu sebagai berikut:

1. Himpunan Kandidat
   Himpunan kandidat merupakan semua ruang sampel yang dapat dipilih sebagai solusi.
2. Himpunan Solusi
   Himpunan solusi merupakan himpunan bagian dari himpunan kandidat yang merupakan solusi dari permasalahan yang ada.
3. Fungsi Seleksi
   Fungsi seleksi merupakan fungsi yang digunakan untuk menyeleksi anggota-anggota dari himpunan kandidat yang akan diperiksa menggunakan fungsi kelayakan agar dapat masuk ke dalam himpunan solusi.
4. Fungsi Layak
   Fungsi layak merupakan fungsi yang digunakan untuk memeriksa apakah solusi yang dipilih merupakan solusi yang layak untuk dimasukkan ke dalam himpunan solusi atau tidak.
5. Fungsi Objektif
   Fungsi objektif adalah solusi optimum yang dihasilkan.

Persoalan optimasi (*optimization problems*) merupakan persoalan mencari solusi optimal. Hanya ada dua macam persoalan optimasi, yakni:

1. Maksimasi (*maximization*)
2. Minimasi (*minimization*)

Game engine dapat ditambahkan pemain baru, mengganti pemain, melakukan turnamen otomatis dengan merubah *file* game-runner-config.json
Konfigurasi permainan, seperti *damage* lava, dan sebagainya dapat dilakukan dengan mengubah *file* game-config.json
Untuk menjalankan permainan, cukup dijalankan *file* .jar yang diberikan di dalam *game engine*.

BAB III
PEMANFAATAN STRATEGI GREEDY

3.1 Mapping Persoalan Permainan *Worms*

Persoalan pada permainan *worms* dapat dipetakan menjadi beberapa aspek yang harus dipenuhi, yakni:

1. Himpunan Kandidat
   Himpunan kandidat pada penyelesaian permainan ini merupakan kumpulan *command* pemain dan musuh yang akan di-*execute*, yaitu *shoot, dig, move, do nothing, banana* dan *snowball*.
2. Himpunan Solusi
   Himpunan solusi pada penyelesaian permainan ini merupakan kumpulan command yang digunakan untuk memenangkan permainan (solusi).
3. Fungsi Seleksi
   Fungsi seleksi yang digunakan menggunakan strategi greedy yang mengimplementasikan scoring dan *minimax.*
4. Fungsi Layak
   Fungsi Layak pada penyelesaian permainan ini adalah ketika melakukan eksekusi tidak menghasilkan error, maka kandidat valid.
5. Fungsi Objektif
   Solusi optimum yang ditawarkan adalah dengan memberikan poin terhadap (*scoring*) command yang dilakukan oleh musuh, lalu meminimasi serta maksimasi kemungkinan langkah serta kondisi dari worms dari pemain dan worms dari lawan.

3.2 Eksplorasi Alternatif solusi yang mungkin dipilih dalam persoalan Worms

Salah satu algoritma yang populer di kalangan *bot* adalah minimax dengan alpha-beta pruning. Algoritma tersebut cukup efisien, bekerja cepat, dan dapat diparalel. Berbagai varian algoritma tersebut mengoptimasi *edge condition*, seperti ronde awal/akhir, efisiensi pencarian, dsb.

Tentunya (hampir) semua permainan dapat diubah menjadi sebuah graf. Graf tersebut dapat ditelusuri menuju kondisi menang. Sehingga, algoritma *traversal* graf secara teknis dapat digunakan.

3.3 Analisis efisiensi dari kumpulan alternatif solusi greedy yang dirumuskan

Algoritma *greedy* memiliki dua komponen kritis, fungsi kandidat dan fungsi seleksi. Fungsi kandidat biasanya menghasilkan puluhan, bahkan ratusan solusi. Sehingga fungsi seleksi harus cepat dan efisien.

Sehingga, kompleksitas waktu total adalah:

$$T(n) = T_s(N_c(n)) + T_c(n)$$

Dengan:

- $T(n)$: Kompleksitas waktu untuk n buah keadaan awal
- $T_s(n)$: Waktu yang dibutuhkan untuk menyeleksi satu kandidat
- $N_c(n)$: Jumlah kandidat perintah untuk satu kondisi awal
- $T_c(n)$: Waktu untuk menghasilkan semua kandidat

3.4 Analisis efektivitas dari kumpulan alternatif solusi greedy yang dirumuskan

Karena jarak prediksi yang terbatas, algoritma *greedy* hampir selalu kalah dengan algoritma yang memprediksi gerakan lawan. Algoritma kami cukup *advanced*, dimana kami berusaha memprediksi **satu** perintah lawan. Tentunya dapat dilakukan prediksi lebih jauh, namun algoritma tersebut bukanlah *greedy* lagi.

3.5 Strategi greedy yang dipilih

Strategi seleksi kandidat dibagi menjadi 2 tahap, yaitu *scoring* dan *minimax*.
Scoring menggunakan beberapa kriteria, diantaranya:

- Setiap *worm* player mati: -1000
- Setiap *worm* musuh mati: +1000
- Setiap *health point* player: +50
- Setiap *health point* musuh: -50
- *Worm* player di atas lava: -1000
- Dirt di sekitar *worm* player: -1
- Setiap *banana*: 20
- Setiap *snowball*: 20
- Unit jarak *euclidean* player dengan musuh terdekat (jika < 2 tidak dihitung): -1
- Player berada horizontal, vertikal, atau diagonal dengan musuh: 50

.

Setelah semua kandidat diberi skor, akan dipilih **kandidat perintah terbaik yang musuh berusaha memperburuk**. Secara formal, adalah sebagai berikut:

$$v = \max(c_p \in H_p, \min(c_o \in H_o, f(c_p, c_o)))$$

Dengan:

- v : skor akhir
- c_p : perintah player
- H_p : Himpunan kandidat pemain
- c_o : perintah musuh
- H_o : Himpunan kandidat musuh
- $f(x, y)$: skor hasil eksekusi perintah

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi program dalam game engine

MoveIterator:
Kelas abstrak ini mengiterasi semua kemungkinan perintah, panggil `iterateMove()` dengan argumen keadaan awal dan selektor yang digunakan. Ia akan mengembalikan perintah terbaik yang dapat ditemukan.

SingleThreadedIterator:
Kelas konkrit ini mengiterasi perintah menggunakan satu *thread*.

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.NoSuchElementException;

public class SingleThreadedIterator extends MoveIterator {

     private static class SingleWormIterator implements
Iterator<Command> {

     private static class CmdMoveIter implements
Iterator<Command> {
          Coord orig;
          Command cmd;
          Coord.Direction dir;

          public CmdMoveIter(Coord from) {
               orig = new Coord(from);
               cmd = new Command();
               dir = Coord.Direction.N;
               Coord c = new Coord(orig);
               c.moveToDirection(dir);
               cmd.setMove(c);
          }

          @Override
          public boolean hasNext() {
               return dir != Coord.Direction.NW;
          }

          @Override
          public Command next() throws
NoSuchElementException {
               switch (dir) {
               case N:
```

```
                    dir = Coord.Direction.NE;
                    break;
               case NE:
                    dir = Coord.Direction.E;
                    break;
               case E:
                    dir = Coord.Direction.SE;
                    break;
               case SE:
                    dir = Coord.Direction.S;
                    break;
               case S:
                    dir = Coord.Direction.SW;
                    break;
               case SW:
                    dir = Coord.Direction.W;
                    break;
               case W:
                    dir = Coord.Direction.NW;
                    break;
               case NW:
                    throw new NoSuchElementException();
               }
               Coord c = new Coord(orig);
               c.moveToDirection(dir);
               cmd.setMove(c);
               return cmd;
          }
     }

     private static class CmdDigIter implements
Iterator<Command> {
          Coord orig;
          Command cmd;
          Coord.Direction dir;

          public CmdDigIter(Coord from) {
               orig = new Coord(from);
               cmd = new Command();
               dir = Coord.Direction.N;
               Coord c = new Coord(orig);
               c.moveToDirection(dir);
               cmd.setDig(c);
          }

          @Override
          public boolean hasNext() {
               return dir != Coord.Direction.NW;
```

```
            }

            @Override
            public        Command        next()        throws
NoSuchElementException {
                  switch (dir) {
                  case N:
                        dir = Coord.Direction.NE;
                        break;
                  case NE:
                        dir = Coord.Direction.E;
                        break;
                  case E:
                        dir = Coord.Direction.SE;
                        break;
                  case SE:
                        dir = Coord.Direction.S;
                        break;
                  case S:
                        dir = Coord.Direction.SW;
                        break;
                  case SW:
                        dir = Coord.Direction.W;
                        break;
                  case W:
                        dir = Coord.Direction.NW;
                        break;
                  case NW:
                        throw new NoSuchElementException();
                  }
                  Coord c = new Coord(orig);
                  c.moveToDirection(dir);
                  cmd.setDig(c);
                  return cmd;
            }
      }

      private    static    class    CmdShootIter    implements
Iterator<Command> {
            Command cmd;
            Coord.Direction dir;

            public CmdShootIter() {
                  cmd = new Command();
                  dir = Coord.Direction.N;
                  cmd.setShoot(dir);
            }
```

```
            @Override
            public boolean hasNext() {
                  return dir != Coord.Direction.NW;
            }

            @Override
            public Command next() throws
NoSuchElementException {
                  switch (dir) {
                  case N:
                        dir = Coord.Direction.NE;
                        break;
                  case NE:
                        dir = Coord.Direction.E;
                        break;
                  case E:
                        dir = Coord.Direction.SE;
                        break;
                  case SE:
                        dir = Coord.Direction.S;
                        break;
                  case S:
                        dir = Coord.Direction.SW;
                        break;
                  case SW:
                        dir = Coord.Direction.W;
                        break;
                  case W:
                        dir = Coord.Direction.NW;
                        break;
                  case NW:
                        throw new NoSuchElementException();
                  }
                  cmd.setShoot(dir);
                  return cmd;
            }
      }

      private static class CmdBananaIter implements
Iterator<Command> {
            WormExt w;
            Command cmd;

            static final int yst = -5;
            static final int endy = 6;
            static final int[] dlx = { 0, 1, -3, 4, -4, 5, -4,
5, -4, 5, -5, 6,
                  -4, 5, -4, 5, -4, 5, -3, 4, 0, 1, };
```

```
            int x, endx, y;

            public CmdBananaIter(Worm worm) {
                  try {
                        w = (WormExt) worm;
                        if (w.getBananaCount() <= 0)
                        throw new ClassCastException();
                        cmd = new Command();
                        y = yst;
                        x = dlx[(y - yst) * 2];
                        endx = dlx[(y - yst) * 2 + 1];
                  } catch (ClassCastException e) {
                        cmd = null;
                  }
            }

            @Override
            public boolean hasNext() {
                  return (cmd != null)
                        && ((y != (endy - 1)) || (x != (endx -
1)));
            }

            @Override
            public        Command        next()        throws
NoSuchElementException {
                  cmd.setBanana(new  Coord(x  +  w.getX(),  y  +
w.getY()));
                  x++;
                  if (x >= endx) {
                        y++;
                        if (y >= endy)
                        throw new NoSuchElementException();
                        x = dlx[(y - yst) * 2];
                        endx = dlx[(y - yst) * 2 + 1];
                  }
                  return cmd;
            }
      }

      private    static    class    CmdSnowballIter    extends
CmdBananaIter {
            public CmdSnowballIter(Worm worm) {
                  super(worm);
            }

            @Override
```

```
            public        Command        next()        throws
NoSuchElementException {
                  Command ret = super.next();
                  Coord c = ret.getTarget();
                  ret.setSnowball(c);
                  return ret;
            }
      }

      Worm w;
      Iterator<Command> subiter;

      enum IterType {
            MOVE, DIG, BANANA, SNOWBALL, SHOOT, DONE
      };

      IterType it;

      public SingleWormIterator(Worm w) {
            this.w = w;
            subiter = new CmdMoveIter(w.getPos());
            it = IterType.MOVE;
      }

      @Override
      public boolean hasNext() {
            return it != IterType.DONE;
      }

      @Override
      public Command next() throws NoSuchElementException {
            if ((subiter != null) && (subiter.hasNext())) {
                  Command cmd = subiter.next();
                  cmd.setWormId(w.getID());
                  return cmd;
            }
            switch (it) {
            case MOVE:
                  it = IterType.DIG;
                  subiter = new CmdDigIter(w.getPos());
                  return next();
            case DIG:
                  it = IterType.BANANA;
                  subiter = new CmdBananaIter(w);
                  return next();
            case BANANA:
                  it = IterType.SNOWBALL;
                  subiter = new CmdSnowballIter(w);
```

```
                return next();
            case SNOWBALL:
                it = IterType.SHOOT;
                subiter = new CmdShootIter();
                return next();
            case SHOOT:
                it = IterType.DONE;
                subiter = null;
                return new Command();
            case DONE:
            }
            throw new NoSuchElementException();
        }
        }

        private static class PlayerIterator implements
Iterator<Command> {
        Worm[] arr;
        int ix;
        Iterator<Command> subiter;

        public PlayerIterator(Player p) {
            arr = p.getWorms();
            ix = 0;
            subiter = null;
        }

        @Override
        public boolean hasNext() {
            return (ix != arr.length) || subiter.hasNext();
        }

        @Override
        public Command next() throws NoSuchElementException {
            if ((subiter != null) && (subiter.hasNext())) {
                return subiter.next();
            }
            subiter = new SingleWormIterator(arr[ix]);
            ix += 1;
            return this.next();
        }
        }

        public SingleThreadedIterator() {
        }

        @Override
```

```
     public Command iterateMove(State state, MoveSelector
selector) {
     HashMap<Integer, Command> cmd = new HashMap<Integer,
Command>();

     try {
          BBox bbox = state.getMap().makeBBox();
          Command bestCmd = new Command();
          State best = (State) state.clone();
          State last = (State) best.clone();
          State cur = (State) best.clone();

          int myPid = state.getMyPlayerID();
          {
               cmd.put(myPid, bestCmd);
               int opponentPid = (myPid == 1) ? 2 : 1;
               Iterator<Command> ito = new PlayerIterator(
                    state.getPlayerByID(opponentPid));
               while (ito.hasNext()) {
                    Command cmdo = ito.next();
                    if ((cmdo.getTarget() != null)
                         &&
!cmdo.getTarget().isBounded(bbox))
                    continue;
                    cmd.put(opponentPid, cmdo);
                    cur.copyFrom(state);
                    try {
                    CommandExecutor.execute(cur, cmd);
                    if (selector.isStateBetter(best, cur)) {
                         State temp = cur;
                         cur = best;
                         best = temp;
                    }
                    }                                   catch
(CommandExecutor.InvalidCommandException e) {
                    }
               }
          }

          Iterator<Command> itp = new SingleWormIterator(
                    state.getPlayerByID(myPid)

.getWormByID(state.getCurrentWormID()));
          while (itp.hasNext()) {
               last.copyFrom(state);
               Command cmdp = itp.next();
               if ((cmdp.getTarget() != null)
                    && !cmdp.getTarget().isBounded(bbox))
```

```
                        continue;
                    cmd.put(myPid, cmdp);
                    boolean isMoveValid = false;
                    int opponentPid = (myPid == 1) ? 2 : 1;
                    Iterator<Command> ito = new PlayerIterator(
                        state.getPlayerByID(opponentPid));
                    while (ito.hasNext()) {
                        Command cmdo = ito.next();
                        if ((cmdo.getTarget() != null)
                            &&
!cmdo.getTarget().isBounded(bbox))
                        continue;
                        cmd.put(opponentPid, cmdo);
                        cur.copyFrom(state);
                        try {
                        CommandExecutor.execute(cur, cmd);
                        isMoveValid = true;
                        if (selector.isStateBetter(last, cur)) {
                            State temp = cur;
                            cur = last;
                            last = temp;
                        }
                        }                                    catch
(CommandExecutor.InvalidCommandException e) {
                        }
                    }
                    if (!isMoveValid)
                        continue;
                    if (selector.isStateBetter(last, best)) {
                        bestCmd           =           (Command)
cmd.get(myPid).clone();
                        State temp = best;
                        best = last;
                        last = temp;
                    }
                }

                return bestCmd;
            } catch (CloneNotSupportedException e) {
                throw new RuntimeException(e);
            }
            }

}
```

MoveSelector:
Kelas abstrak ini mengurutkan state-state hasil, menggunakan metode `isStateBetter()`

GreedySelector:
Kelas konkrit ini menyeleksi state menggunakan *scoring*. Ini adalah inti dari algoritma *greedy*.

```
public class GreedySelector extends MoveSelector {

      public GreedySelector() {
      }

      /*
      * Create score for the state
      */
      private double scoreState(State state) {
      double score = 0.0;

      int myPid = state.getMyPlayerID();
      int opponentPid = (myPid == 1) ? 2 : 1;

      Player myPlayer = state.getPlayerByID(myPid);
      for (Worm w : myPlayer.getWorms()) {
            if (w.getHealth() <= 0)
                  score -= 1000.0;
            score += w.getHealth() * 50;
            Map map = state.getMap();
            if         (map.getCell(w.getPos()).type         ==
Map.CellType.LAVA)
                  score -= 1000.0;
            Coord pos = new Coord(w.getPos());
            for (int x = -1; x < 2; x++)
                  for (int y = -1; y < 2; y++) {
                        try {
                        Map.Cell cell = map.getCell(pos.getX() +
x,
                                    pos.getY() + y);
                        if (cell.type == Map.CellType.DIRT)
                              score -= 1;
                        } catch (IndexOutOfBoundsException e) {
                        }
                  }
            try {
                  WormExt we = (WormExt) w;
                  score += we.getBananaCount() * 20;
                  score += we.getSnowballCount() * 20;
            } catch (ClassCastException e) {
            }
      }
```

```
      Player                  opponentPlayer                  =
state.getPlayerByID(opponentPid);
      for (Worm w : opponentPlayer.getWorms()) {
            if (w.getHealth() <= 0)
                  score += 1000.0;
            score -= w.getHealth() * 50;
      }

      for (Worm w : myPlayer.getWorms()) {
            if (w.getHealth() <= 0)
                  continue;
            Coord posw = w.getPos();
            Worm to = null;
            for (Worm w2 : opponentPlayer.getWorms()) {
                  if (w2.getHealth() <= 0)
                        continue;
                  Coord pos = w2.getPos();
                  if ((to == null)
                        ||          (posw.distance(pos)          <
to.getPos().distance(posw))) {
                        to = w2;
                  }
            }
            if (to == null)
                  continue;
            Coord posw2 = to.getPos();
            double dist = posw2.distance(posw);
            assert dist > 0.0;
            if (dist >= 2)
                  score -= dist * 8;
            int dx = posw.getX() - posw2.getX();
            int dy = posw.getY() - posw2.getY();
            if ((dx == 0) || (dy == 0) || (dx == dy) || (dx ==
-dy))
                  score += 50;
      }

      return score;
      }

      /*
      * Return true if state is better
      */
      @Override
      public boolean isStateBetter(State state, State other) {
      double thisScore = scoreState(state);
      double otherScore = scoreState(other);
      return thisScore > otherScore;
```

```
            }

        }
```

4.2 Struktur data

Untuk menyimpan keadaan permainan, dibuatlah kelas keadaan `State`. Kelas ini menyimpan peta, daftar pemain (yang berisi daftar worm), dan konfigurasi permainan.

Penyimpanan peta menggunakan kelas `Map`. Kelas ini berisi *array* integer yang mengkodekan keadaan setiap sel. Pemrosesan kode ini dibantu oleh subkelas `Cell`, yang mengabstraksikan keadaan suatu sel.

4.3 Analisis desain solusi algoritma greedy

Solusi kami mirip dengan algoritma **minimax**, yang berusaha memaksimumkan hasil yang diminimumkan lawan. Tentunya minimax melakukan prediksi untuk beberapa ronde, sedangkan solusi kami hanya melakukan satu ronde.

Untuk *scoring*, masih diperlukan beberapa *tweaking*. Namun untuk sekarang *bot* kami dapat melakukan tugasnya.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

- Algoritma greedy merupakan algoritma yang bisa digunakan dalam memecahkan persoalan mencari solusi optimal, yaitu persoalan maksimasi dan minimasi.
- Algoritma greedy merupakan algoritma yang memecahkan persoalan secara langkah per langkah, dimana di setiap langkah akan diambil pilihan terbaik saat itu, tanpa melihat konsekuensi yang ada kedepannya.
- Solusi yang didapatkan dari penggunaan algoritma greedy belum tentu merupakan solusi optimum (terbaik), hal ini dikarenakan algoritma greedy memiliki pilihan fungsi seleksi yang berbeda-beda, sehingga pemilihan fungsi seleksi yang tepat sangat penting dalam penggunaan algoritma greedy.
- Algoritma greedy dapat diimplementasikan sebagai strategi untuk bot dalam permainan Worms.

5.2 Saran

Spesifikasi tugas besar yang diberikan sedikit kurang detail mengenai bagian program mana yang sebenarnya harus diubah-ubah atau diberi blok program baru, sehingga menimbulkan kebingungan dalam pengerjaan tugas besar ini, maka kami menyarankan agar spesifikasi tugas besar dibuat lebih detail.

DAFTAR PUSTAKA

Github: Entelect Challenge/2019-Worms. Diakses pada 10 Februari 2021. https://github.com/EntelectChallenge/2019-Worms/blob/develop/game-engine/game-rules.md

“Algoritma Greedy (Bagian 1)”. Informatika.stei.itb.ac.id. Diakses pada 13 Februari 2021. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf

“Algoritma Greedy (Bagian 2)”. Informatika.stei.itb.ac.id. Diakses pada 13 Februari 2021. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf

“Algoritma Greedy (Bagian 3)”. Informatika.stei.itb.ac.id. Diakses pada 13 Februari 2021. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag3.pdf