Final Report - Group 21

Trash Intelligent Recyclers

Diogo Lopes
AASMA
Instituto Superior Técnico
Lisbon, Portugal
diogo.andre.fulgencio.lopes@tecnico.ulisboa.pt

Gonçalo Mateus
AASMA
Instituto Superior Técnico
Lisbon, Portugal
goncalo.fililpe.mateus@tecnico.ulisboa.pt

Nuno Estalagem
AASMA
Instituto Superior Técnico
Lisbon, Portugal
nuno.estalagem@tecnico.ulisboa.pt

ABSTRACT

The problem we chose to approach in this project is where a group of two types of intelligent robots, detectors and collectors, are used to detect and recycle trash that is distributed all around a certain area. The objective is to gather trash and recycle it as quickly and efficiently as possible.

1 Introduction

Park littering is an issue nowadays. People tend to dispose of their trash inappropriately, which not only affects nature drastically, but also makes individuals less keen on using the park. This way, in our project, we developed a **multi-agent system**, in which **agents** cooperate and find the **most efficient** way of recycling litter in a park. Given a **number of two types of robots** that are spawned all around a park according to several approaches, these robots have to **recycle the found trash** in its respective container. The number of containers is predefined and all robots will know their positions. The two types of robots we developed are **detectors** which are robots whose main objective is to **detect trash** and **collectors** that are robots that **collect the trash** and deliver it to the respective recycling container. One collector is limited to carrying a **specific amount** of trash and **must decide**, along with the remaining collectors, who will end up taking the trash to its respective container.

2 Environment

The created environment is a **littered park** that is divided **into 6 sectors**. Each sector is given a **probability**, which influences the **amount** of **trash spawned** in it. For each sector, this probability is defined **before running** any of the **implemented approaches** and is set **randomly** from a **list** of **predefined probabilities**.

The park is **composed** of **60x32 tiles**, each one of them **being** a **16x16 pixel square**. A **sector** is **constituted** by **20x16 tiles** i.e, **one sixth** of the **entire park**, and is visually **delimited** by a **white grid**. A **Robot/Trash** is **represented** by 1 **single tile**.

In the created **littered park** it's possible to find **6 containers** (2x2 dark blue squares) and their **corresponding Field Of View** (light blue), in which the **collector robots deliver** the found **trash**. **Robots** are, therefore, **not allowed** to **trespass** any **container**, only its **Field Of View**.

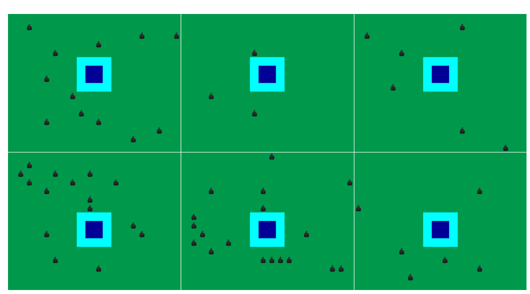

Figure 1 - Example of environment

- Containers
- Containers field of view
- Spawned trash

Figure 1 makes understanding the trash spawn probabilities easier; **Sectors 1, 4 and 5** clearly have **larger trash spawning odds** than the remaining sectors.

Figure 2 – Trash representation

The environment for this project is **inaccessible**, meaning robots **do not** have **complete knowledge** of the environment, can **only see** what is **around them** up to a **certain range** and **have knowledge** of **each container's location**.

The developed environment is **dynamic,** since it is modified while other robots are deliberating. From a robot's point of view, **trash** may be **spawned**/**collected** and other robots **can move** while he is **yet** to make a **decision**.

It is also **non-episodic**, since the collected trash influences the environment's future, thus it cannot be separated into independant intervals.

3 Agents

The available agents are robots that **act in an independent way**, while being able to interact between them. The different types of developed agents are **detectors** and **collectors**, each having different sensors and actuators.

Detectors

The Detector's main goal is to detect trash through its larger field of view and broadcast these locations to the Collectors.

There are t**wo available approaches** for these robots, the first one being a **fixed constant sweep** through the environment and the other one a **sector sweep**, based on the **detector's belief** of the **sector** with the **biggest amount** of **trash**.

While doing any of the specified movements, two detectors will not collide, and instead detect the proximity and change its trajectory to avoid collision.

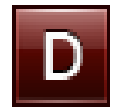

Figure 3 – Detector's representation

For this to be accomplished the following sensors and actuators were assigned:

- **Sensors**
    - Detecting trash in a wide radius (5x5 tiles field of view);
    - Detecting other robots' proximity.

- **Actuators**
    - Sweep across the environment;
    - Move towards specific sectors;
    - Send messages to collector robots when detecting trash.

Collectors

The Collector's main **goal** is to **collect trash** through its **small field of view** or **move towards trash** that was **encountered** by the **Detector** robots.

To collect trash, the collector must **move** to the **exact position** where the **trash** is **located**. A **collector's** storage is set to **4**, meaning he can **carry up** to **4 bags** of trash.

The collectors are also able to **deliver** the **stored trash** to the **containers**, by **moving towards** a **container's Field Of View**.

While **doing** any of the **specified movements**, two collectors will **not collide**, **changing** its **trajectory** to **avoid collision** instead.

Figure 4 – Collector's representation

For this to be accomplished the following sensors and actuators were assigned:

- **Sensors**
  - Detect trash in a small radius;
  - Detect other robots' proximity;
  - Receive messages from both Collectors and Detectors.
- **Actuators**
  - Move randomly around the environment;
  - Move towards a specific trash;
  - Move towards a container to deliver the collected trash;
  - Grab/Drop trash;
  - Send messages to other Collectors.

The collectors must find the trash after being noticed by the detectors and move it to the containers. This process should be done in the fastest approach possible.

Movement Implementation of the Agents

The **specified** types of **agents** can move in a **random** way or **towards** a **specific** tile, being for this implemented **different approaches**.

For a **random movement**, the agent makes a **random** choice from a **list** of **directions**, i.e **North**, **South**, **East**, **West**. If this **movement is possible** then the **agent moves towards** the **corresponding** tile. Otherwise, the choice is **redone**. In addition, if there are **no possible movements**, the robot will **stay** in the **same position** and **wait** for the **next turn** to **choose** a movement.

In case of **moving** towards a **specified tile**, an **A* search algorithm** was implemented, guaranteeing **the most efficient path** between **two tiles** in the **park environment**. When a **robot** is on the **verge** of **colliding** with **another**, he **recalculates** his **path** in order for the **collision** to be **avoided**.

The **heuristic utilized** by the **implemented A* search algorithm** consists in **comparing** the **f_costs** of a certain point's **neighbors**, up **until** the **optimal path** to the **requested point** is **discovered**. More specifically, a **point's f_cost** consists in **summing** its **distance** from the **starting point** (**g_cost**) with the **distance** to the **end point** (**h_cost**).

The A*search **begins** by **exploring** the **starting point** and **adding all** the **neighbors** to which the robot **can move** from **there** to a **"to be explored" list**. This happens at the beginning, since the **algorithm verifies** each of a **currently explored point's neighbors** and appends the ones to which the r**obot can move** and either have **not been explored** previously, or whose **g_cost** is **lower** than its **currently** assigned **g_cost** (this means that the A* search found a **cheaper way** of **accessing** that **point** from the **starting point**) to the **"to be explored list"** and **calculates/updates** the **neighbor's** current **g_cost, h_cost** and **f_cost**.

The A* search then **compares** all **neighbors f_costs**, **chooses** the one with **either the lowest f_cost** or, in case two neighbors have **equal f_costs**, the one whose **h_cost** is **inferior,** and **repeats** the **same process** (**verifying** the **neighbors points**, **appending** them to the to be **explored list**, potentially **updating** their **costs…**) until the **end point** is **reached**. In that scenario, the **algorithm returns** the **obtained path** between the **starting point** and the **end point**.

4 Architectures / Decision Making

In the project, **4 architectures** were **developed**, this way it's possible to **compare** the **results** and **conclude** their **advantages** and **disadvantages**.

Figure 5 - Available architectures

The user can **choose** the **desired architecture** to be run in the **project simulation** by **selecting** one of the **buttons** shown in Figure 5 and will be able to **observe** the **different robots behaviors** in each one of them. After the simulation is run, a pop up will appear and the user should re-run the script if has the desire to simulate again.

Random Collectors

This architecture is the **simplest** one and was used mainly to **test** the **environment**, robots **basic movement** and **interactions**, i.e **picking** up and **delivering trash** to a **container**.

There are only **3 collector robots** in the **environment**, each of which moves in **random** directions (North, South, East, West), while also successfully **avoiding collisions**.

These collectors **don't** have any **information** of the **environment**, only their **current position**, and are only able to **pick up/deliver trash** when in its exact **same location** or in the **container's field of view**, respectively.

Reactive Collectors

In this architecture, Collectors **demonstrate** a **reactive behavior** towards what is **detected** in their **field of view** (3x3 tiles). This way, it's possible to better **understand** each **Collector's decision** when encountering **trash** or a **container field of view** in its **own field of view**.

During the simulation, 3 collector robots **move randomly** throughout the environment until **encountering** a **trash/container** in its **field of view**. Upon this happening, some **conditions** are taken into **consideration**:

- if the **collector** has **trash** to **deliver** and **has** a **container** in its **field of view**, calculates **A* search** to **desired destination** and **delivers trash**;
- else if the **collector** still has **available storage** and **detects** a **trash** in its **field of view**, calculates the **A* search** to the **desired destination** and **picks up** the **trash**.

The Collectors used in this architecture **only** have **information** from **what they see** in their **field of view**.

Intelligent Collectors

Unlike all previous architectures, Collectors **communicate** with **each other** to **decide** which one **picks up** the **detected trash**. In this prototype, **trash** is **detected** either by **Collector** or **Detector Robots**. **Detectors** do **not pick up trash**, but **broadcast** its existence instead.

This prototype makes use of **2 detectors**, whose **movement** consists in an **environment sweep** i.e, each **detector** is given **half** of the **park** to **sweep** (social conventions). Therefore, a **Detector** moves in **specific directions** to **cover** the **half** that was **assigned** to him.

Nonetheless, the **detectors** make use of **A* search** to move **between specific points** of their **clearly defined path**, meaning that **collisions** with **other robots** are **avoided**, since it **recalculates** its **path** to the **desired point** in case another **robot** comes **across** his **path**.

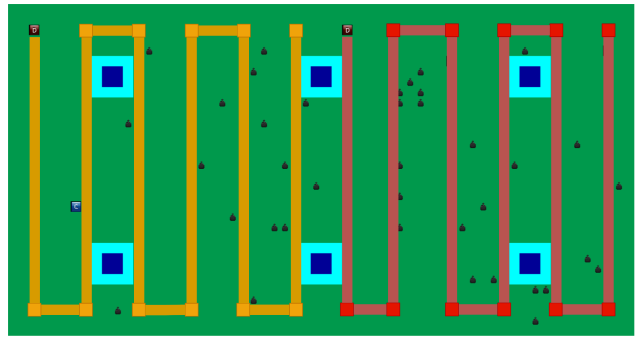

Figure 6 - Detectors predefined sweep

This **architecture's environment** contains **3 Collectors**, which spawn in **random locations**.

When Detectors **broadcast** a certain **trash's existence**, the **respective trash** is appended to a "**to be discussed by the Collectors" list**.

Each Collector has a **variable** "myTrash" that **stores** the **trash supposed** to **collect**. This decision is made **based** on the **trash** that is the **closest** to a certain **Collector**.

Collectors "act" by their **desires**. If **more** than **half** of their **storage is available** they will want to **pick up trash**. On the other hand, if a **Collector's storage** is at least **half full**, his **desire** will be to **deliver** the **trash** it contains.

When the **Collector's** desire is to **pick up** trash he **calculates**, in **each step**, the **trash** to which he is the **closest one to** and, in case he is the **Collector** that is the **closest** to that **trash**, he **assigns** it to **himself**.

The **path** between a **Collector** and the **trash** to him assigned, as well as the **route between** a **Collector** and a **Container's Field Of View**, are both **defined** by the **A* search algorithm** explained **above**.

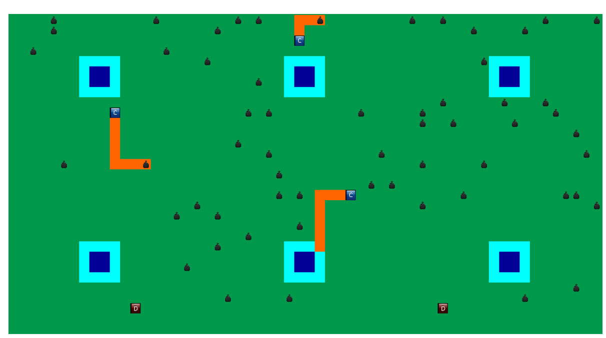

Figure 7 - Collectors performing A* search

There are some **conditions** to be **taken** into **consideration** in regard to the **Collector's movements**. A Collector will:

- **Move randomly** when its storage is **empty** and no trash has been **assigned** for it to **pick up** i.e, a robot will **move randomly** if it "has **nothing to do**";

- Make a "**switch**" in case the **trash** he is the closest to **differs** from the one **already assigned** to him: the previously assigned trash is **put back** in the "**to be assigned list**" and the "**new closest trash**" is **assigned** to the **Collector**;

- **Change** his **path** while moving **towards** a **container** in order to pick up **more trash** if storage is **not totally full**. This case scenario will happen due to the fact that, not only is he the one that is the **closest** to the **referred trash**, but the **container** is also **further away** from **him** than the **rubbish**;

- **Not pick up trash** if its **storage** is **full**, and move **directly towards** the **closest container Field of View**. Nonetheless, it still **detects** the **existence** of **that trash** and **adds** it to the "**to be assigned list**" despite **not being able** to **pick it** up.

The **Detectors** used in this architecture have **knowledge** of **their position**, **field of view** and the **points** they have to **travel** to, in order to **complete** the **environment sweep**.

The Collectors have **knowledge** of **their position**, **field of view**, **container locations** and **detected trash locations**. This way, they are able to easily **cooperate** when **deciding** the **assigned trash** and **calculate** an **A* path** to the **closest container** when they have the **desire** to **deliver** garbage.

Sector Prototype

In this architecture, the **Collector** Robot's **algorithm** is the **exact same** as the **Intelligent Collector** one, meaning that their **reactions** and **decisions** will be the **same** as the ones in the **previously explained environment**.

Therefore, all **Collectors** have **roles** and **collaborate** in order to **collect** and **deliver** the **detected** and **assigned trash** in the **most efficient** way possible.

Unlike the **previous environment**, in which they had **predefined paths** to **sweep**, Detectors will now **collaborate** and move **around** the **park**, sweeping the **sectors** they **believe** to **have** the **most trash (hybrid)**.

Trash spawn is **based** on a **predefined probability list**. **Before running** the **simulation**, each **sector** is **randomly assigned** to one of the 6 **probabilities** from the **predefined** probability **list**. Then, a **random number** of **trash** (between 50 and 80) is **spawned** and **distributed** by each **sector** according to the **probabilities assigned** to them. After selecting the desired architecture, **trash** is **spawned** every **15 frames**. Each sector will have **more** or **less** trash spawned according to their trash **spawn probability**.

When choosing which sector to sweep, Detectors either **stick** by their **belief** and sweep the **sector** whose trash **spawn ratio** is the **highest**, or decide to **sweep** a **random sector**. The probability of **sweeping** a **random sector starts** at **100%** and **decreases** as detectors sweep **more sectors** up to a **minimum** of **5%**. Two Detector robots **cannot sweep** the same sector **simultaneously**, which **avoids inefficient resource management**.

A **chosen sector** is "**locked**", meaning only the **assigned Detector** (and all Collectors) will be **able** to **enter** it, which **prevents** other **Detectors** from **trespassing** a **sector** that is currently **being swept**. Those Detectors **move around** the **locked sector** to **get** to their **destination point** instead.

After choosing a sector to sweep, a Detector will **perform** an **A* search** to obtain the **path** to the **chosen sector's closest corner**. The A* search also has in **consideration** the sectors that are **currently chosen**. Therefore, the calculated path **always prevents** Detectors from **crossing sectors** that are **being swept**.

When arriving at the desired corner, the Detector will **sweep** the **sector**, which can be done in **4 ways depending** on the **corner** (left->right, right->left, bottom->top, top->bottom).

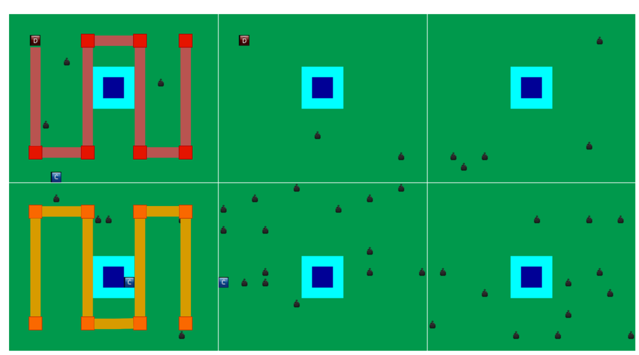

Figure 8 - Detectors possible sector sweep paths
(the other possibilities are the reverse path)

As mentioned, the **process** of **choosing** which **sector** to **sweep** is either done in a **random fashion** or according to the **Detector's beliefs**. A Detector Robot's **sector belief** is defined by the following **expression**:

sector_belief = trash_detected / number_of_times_visited

After a sweep, a **Detector adds** the **number of trash bags** he detected **during** the **sweep** to the **total number of trash detected in that sector**, **increases** the **number** of **times** that sector **has been visited by 1** and makes use of the **formula above** to **update** the **swept sector's belief**.

This way, Detectors will have **more interest** in visiting **sectors** with a **higher sector_belief**. On the other hand, we must **not forget** they also have a **minimum 5% "random probability"** of choosing **any other sector**, which means that a sector **always** has a **chance**, even if **minimal**, to be **swept**.

In this architecture, **Detectors** have **knowledge** of their **position**, **field of view** and **sector corners**, in order to be **able** to **travel** to the **sector** and perform the **sweep**.

On the other hand, Collectors **knowledge remains unchanged** from the **Intelligent Collectors** prototype.

5 Comparative Analysis

In order to make a **comparative analysis** we decided to set **2 scenarios**, where the **frequency** of **spawning trash** varies.

For **each scenario**, **20 runs** of each prototype were performed, being **each run a 60 seconds simulation**. With this, it was possible to obtain average values for the **number of steps** and the quantity of **detected**, **collected** and **delivered trash**, which is very useful for a comparative analysis of the presented architectures.

First Scenario

In the **first scenario trash** is **spawned** every **15 frames according** to each **sector's** spawning **probabilities**.

The following **graphs represent analysis** of various aspects tested and are very **useful** for a **better comparison** between the **developed prototypes**.

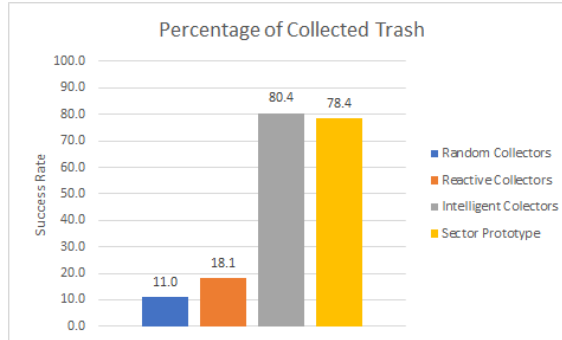

Figure 9 - Percentage of Collected Trash First Scenario

According to this Bar Graph, the **"Intelligent Collector"'s prototype** is the **most efficient** one regarding **trash collecting**. The shown success rates were obtained by dividing the average number of spawned trash in each prototype with the respective average of collected trash.

All prototypes have a **very similar spawned trash average**. Therefore, we can conclude that, when trash is spawned every 15 frames, the Intelligent Collector's prototype is the most effective one.

Keeping in mind that the **"Sector Prototype"'s** percentage of **collected trash** is **very close** to the **"Intelligent Collector"'s**, and knowing that both made use of **Collector Robots with the same algorithm**, we conclude that the **difference** between both architectures can be **justified** by the **dissimilarities** in their **Detector Robot's algorithm**.

The **low success rate** obtained in the **first 2 architectures** was expected, since the **Collectors** are entirely/mostly performing **random actions**, **without** much/any **deliberation or reaction**. Still, the **"Reactive Collector"'s** prototype's success rate was **higher** comparing it to the **"Random Collector"'s** since its **Collector Robots** had a **reactive approach** towards what was in their **FOV**.

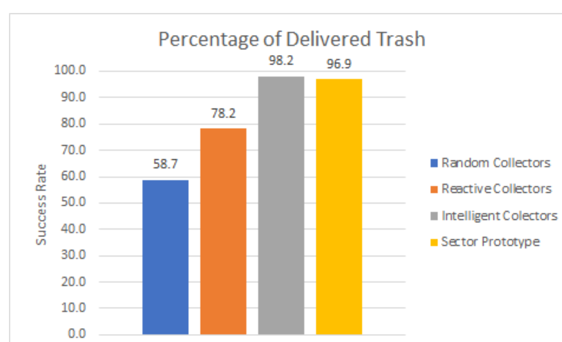

Figure 10 - Percentage of Delivered Trash First Scenario

The bar graph above, which refers to the **percentage average of collected trash that ended up being delivered**, is in line with the first bar graph; both the **“Random Collector”’** and **“Reactive Collector”’s** prototypes have **very little deliberation/reasoning**. Therefore, they are only capable of **delivering trash** in case they **accidentally reach a container’s FOV** or **find** it in their **own field of view by accident**.

On the other hand, our **3rd and 4th prototype** make use of the **same hybrid agent** i.e an **Intelligent Collector**, which uses **A*search** to **deliver trash** in the **most efficient path possible**.

Still, for a **very similar average of spawned trash**, the **Intelligent Collector’s success rate** is **higher**, which leads us to the same **conclusion** as before: the **effectiveness** of each prototype’s **detector approach** is the **reason** behind the **success rate dissimilarities**.

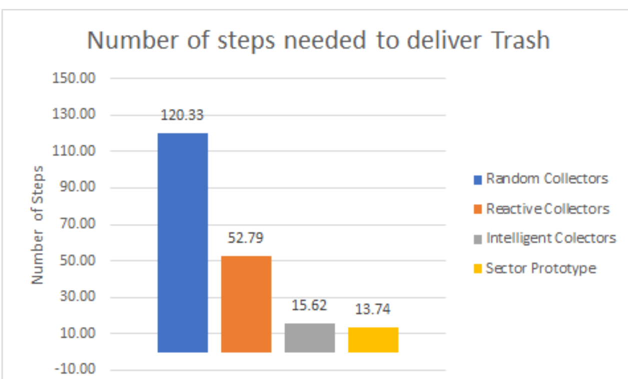

Figure 11 - Number of steps needed to deliver one trash First Scenario

The previous graph is related to each architecture's robot efficiency. A **very efficient architecture** implies that very **few steps** are needed to **deliver a trash bag**. In corroboration with all previous graphs, we came once again to the **conclusion** that the **first two architectures**, due to their **low/inexistent deliberation/reaction/deduction** are extremely **inefficient**; it is **not desirable** for a robot to have an **average** of approximately **120 steps** to deliver a **single trash** bag.

Nonetheless, since the last two prototypes utilize **“Intelligent Collectors”** to **pick up** and **deliver** trash, their **paths** will **always** be **optimal**, which justifies the **high efficiency**.

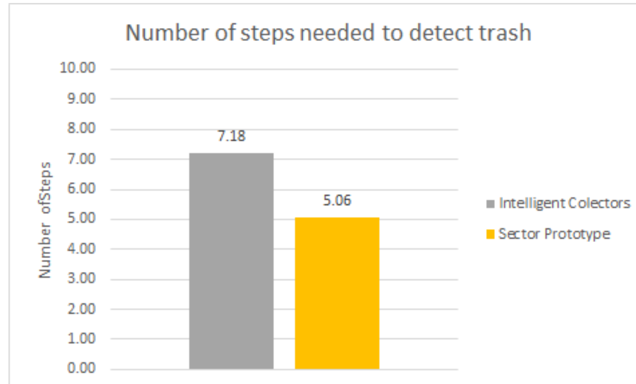

Figure 12 - Number of steps needed to detect one trash First Scenario

The above bar graph has the purpose of **comparing Detector efficiency**, meaning that only the 3rd and 4th prototype were used for stat comparisons. Each prototype’s **result** was **obtained** by **dividing** the **average number of steps made by the detector** with the **average of trash detected by the Detector robots**.

The Detector’s approach for the **Sector Prototype** was surprisingly **more efficient** in comparison to the Intelligent **Collector’s prototype’s** approach. The cause of this efficiency must be related to the way Detector Robots handle their sweeps in the 4th prototype i.e, **instead** of **sweeping half of the environment**, they **only sweep** a **randomly chosen/previously deliberated sector** at a time, with the **odds** of this sector being **randomly chosen** getting **progressively decreased** as **Detectors gain more knowledge** regarding the **trash** in each **sector**.

Thus, by moving **towards** a **chosen sector** and, therefore, only **sweeping** ⅛ of the **environment** at a time while also **guaranteeing** that **most** of the **sectors** swept are the ones corresponding to the **detectors highest belief**, we can justify the **lower number** of **steps** needed to **detect** trash in the **“Sector Prototype”**.

This can also be the reason why the **“Sector”’s Prototype** has a **lower average** of **steps** needed for **trash delivering**; they go **more often** to the **sectors** in which their **belief** is the **highest**, meaning **collectors** tend to be **closer** to those **sectors**.

Second Scenario

In the second scenario trash is **spawned** every **7 frames** according to each **sector's spawning probabilities**, meaning this scenario has a **higher spawning rate** than the **previous** one.

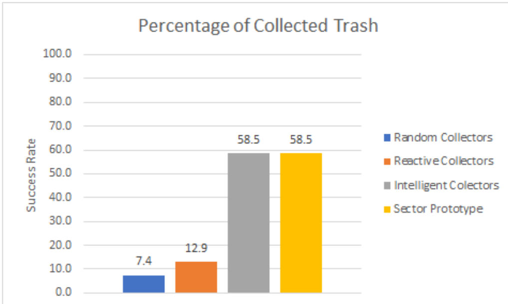

Figure 13 - Percentage of Collected Trash Second Scenario

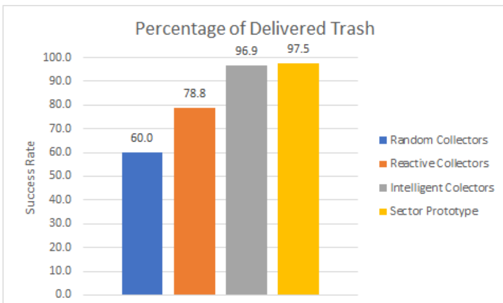

Figure 14 - Percentage of Delivered Trash Second Scenario

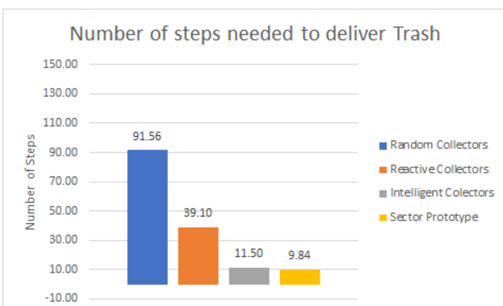

Figure 15 - Number of steps needed to deliver one trash Second Scenario

Figure 16 - Number of steps needed to detect one trash Second Scenario

Since in **this scenario** the amount of **trash spawned increased**, the **first** and **second prototype** were expected to have a **worse percentage of collected trash** (Figure 13), since the number of trash spawned increased along with the chances of a robot finding trash to pick up; **Collectors** definitely **found more trash than previously**, but the **percentage** they were **able to collect** was **lower,** as their movement is (mostly) random and they were not able to find a vast amount of spawned trash.

Figure 14's graph is **similar** to the one in **Figure 10**, since the **Collector Robots** delivering **behavior is the same** in both scenarios.

In **Figure 15**, it's possible to observe a **decrease** in the average **number of steps needed to deliver trash**. Since **trash spawn** has **increased**, its **chance** of **appearing near a container** was **higher**, resulting in a decreased number of steps to perform a delivery.

The graph in **Figure 16** is **identical** to the one in **Figure 12**. We **conclude** that the **"Sector Prototype"** architecture is **more efficient** in detecting trash, even with increased spawns. A **higher efficiency** in **trash detection** does **not** make **higher trash collecting efficiency** a necessary conclusion, as the **percentage** of **collected trash** shown by **Figure 13** is **equal** in both the **3rd** and **4th prototype**.

6 **Conclusions**

Our **analysis** led us to the **conclusion** that both the **Sector** and **Intelligent Collectors prototypes** make the **best usage** of the **resources provided**. **Random** and **Reactive Collector prototypes** were way **less efficient** in comparison, as not only do they **require more steps** to **collect** one **trash**, but **also pick up** way **less** trash **than** the **other** two **prototypes** for the same amount of **time** to act.

Considering the **advantages** that the **3rd** and **4th prototype** have to offer, we came to the **conclusion** that the **Intelligent Collectors** prototype is **more suitable** in situations where **time** is an **issue** and **trash** must be **collected** as **soon** as possible. Nonetheless, the **"Sector Prototype"** is more **adequate** when **resource management** (ex: battery, tires, etc..) is a **concern**, since it does **not** fall behind the **Intelligent Collector's** prototype by much in terms of **trash delivery**, and is way more **efficient** in regards to the **steps** made by its **Detectors**. One way of **reinforcing** these conclusions would be implementing **robots** with **fuel storage** in future work. Another improvement would be allowing Collectors to **pick up trash** to which they are **not** the **closests** to. This would happen in situations where the **trash's closest collector** already has **another** trash **assigned** to him, plus the **total** amount of **steps taken** for it to pick up both the **assigned** garbage and the **trash** the other robot was **not allowed** to collect, is **larger** than the **steps needed** by the **disallowed collector** to **pick up** the **rubbish** in question.