1. **Abstract**

This report explores the development of a critter simulation project, which revolves around crafting diverse critter entities that showcase distinctive behaviors within a virtual realm. The primary emphasis is on characterizing the traits and actions of critters, encompassing bears, tigers, giants, white tigers, and NinjaCats. The report offers an overview of the project's objectives, design, development journey, and outcomes. It further delves into dissecting the simulation's performance and culminates with reflections on potential future enhancements.

1. **Introduction**

Critter simulations present captivating experiments where an array of critters coexist and engage within a virtual ecosystem. These simulations provide valuable insights into various animal behaviors, ecological dynamics, and the repercussions of interspecies interactions. Within the confines of this project, the focus is squarely on the creation of a cadre of critters, each boasting unique traits, all brought to life using Java programming. Each critter class takes the form of a Java class, collectively constituting a vibrant critter ecosystem.

1. **Project Description**

The project's core objective is to create a critter ecosystem in which critters display distinct behaviors aligned with pre-established rules. The simulation layout is delineated as a grid, wherein critters moves within their confines, reacting and adapting to their immediate environment. The project unfolds with the creation of five critter classes: Bear, Tiger, Giant, WhiteTiger, and NinjaCat. Each critter class is endowed with unique behaviors and traits in accordance with the project's specifications.

These critters traverse the grid, engaging in interactions among themselves and dynamically adjusting their actions to suit the context. The simulation relies on the application of predefined rules dictating critter actions, including actions like infection, hopping, or turning. Collectively, these critters form a dynamic ecosystem wherein their interaction fosters a virtual ecosystem.

In this report, we delve into the design and development of these critter classes. We navigate the inner workings and logic governing the program's flow, and we offer an in-depth exploration of the results arising from critter interactions within the simulation. Furthermore, we uncover the implications of critters' behaviors, the ripple effects of their interactions, and the valuable lessons distilled from the project. The report culminates with considerations regarding potential future enrichments for the critter ecosystem simulation.

It's essential to note that this project does not involve creating the application with a main method but, rather, centers around the construction of a spectrum of objects (critters). The result is a simulation that models critter behaviors and interactions in a within the map.

1. **Project Design**

The project design for the critter simulation system is structured around creating a dynamic ecosystem where various critters exhibit distinct behaviors and interact within a virtual environment. To achieve this, we've developed five key critter classes: Bear, Tiger, Giant, WhiteTiger, and NinjaCat, each with its unique traits and actions.

* 1. **Bear Class**

|  |  |  |
| --- | --- | --- |
| **Purpose** |  | The Bear class is responsible for defining the behavior of the Bear critter within the simulation. Its specific purpose is to alternate its visual representation between a slash ('/') and a backslash ('') character. |
| **Method** |  | getColor(): Returns the color of the Bear, which is always black.  toString(): Determines the visual representation of the Bear, alternating between '/' and ''.  getMove(CritterInfo info): Defines the movement logic for the Bear. The Bear hops in one direction unless blocked by a wall or another critter. |
| **Control Structure** |  | The Bear's behavior is straightforward. It continuously alternates between two visual representations, '/', and ''\”. |
| **Input Requirements** |  | The Bear class takes no input parameters for initialization. |
| **Output Results** |  | When executed, the Bear will continuously alternate its appearance between '/' and '', and it will hop forward when conditions permit. |

* 1. **Tiger Class**

|  |  |  |
| --- | --- | --- |
| **Purpose** |  | The Tiger class defines the behavior of the Tiger critter within the simulation. Its specific purpose is to change direction based on its surroundings to avoid obstacles and maintain a sense of companionship with fellow Tigers. |
| **Mathod** |  | getColor(): Returns the color of the Tiger, which is always orange.  toString(): Determines the visual representation of the Tiger, which depends on its direction and behavior.  getMove(CritterInfo info): Defines the movement logic for the Tiger. Tigers make decisions based on their perception of the surroundings, such as whether there's a wall, another Tiger, or an enemy critter in their path. |
| **Control Structure** |  | The Tiger's control structure includes decision-making based on the perceived environment. If there's an obstacle (wall or another critter) ahead, the Tiger makes a left turn. If there's a fellow Tiger, it makes a right turn. If an enemy is in front, it continues forward to potentially infect the enemy. |
| **Input Requirements** |  | The Tiger class takes no input parameters for initialization. |
| **Output Results** |  | When executed, the Tiger will change it’s colour randomly after 3 steps. |

* 1. **Giant Class**

|  |  |  |
| --- | --- | --- |
| **Purpose** |  | The Giant class defines the behavior of the Giant critter within the simulation. Its specific purpose is to cycle through a set of phrases, creating a random element within the ecosystem. |
| **Method** |  | getColor(): Returns the color of the Giant, which is always gray.  toString(): Determines the visual representation of the Giant based on its current phrase.  getMove(CritterInfo info): Defines the movement logic for the Giant. Giants cycle through their phrases and can infect other critters in their path. |
| **Control Structure** |  | The Giant's control structure involves selecting the next phrase in the cycle and making decisions to hop forward or infect enemy critters when conditions permit. |
| **Input Requirements** |  | The Giant class takes no input parameters for initialization. |
| **Output Results** |  | When executed, the Giant will cycle through phrases and interact with other critters by hopping or infecting them. |

* 1. **WhiteTiger Class**

|  |  |  |
| --- | --- | --- |
| **Purpose** |  | The WhiteTiger class defines the behavior of the WhiteTiger critter within the simulation. Its specific purpose is to remain white in color and to infect other critters when an enemy is detected. |
| **Method** |  | getColor(): Returns the color of the WhiteTiger, which is always white.  toString(): Determines the visual representation of the WhiteTiger based on its infection status.  getMove(CritterInfo info): Defines the movement logic for the WhiteTiger. WhiteTigers attempt to infect enemy critters when detected and change direction or hop if conditions require. |
| **Control Structure** |  | The WhiteTiger's control structure involves changing direction, hopping, or infecting enemy critters based on its surroundings. |
| **Input Requirements** |  | The WhiteTiger class takes no input parameters for initialization. |
| **Output Results** |  | When executed, the WhiteTiger remains white in color and attempts to infect enemy critters when detected. It is initially visually represented by “tgr” in white. Once it has infected a critter, it is visually represented by “TGR” in white. |

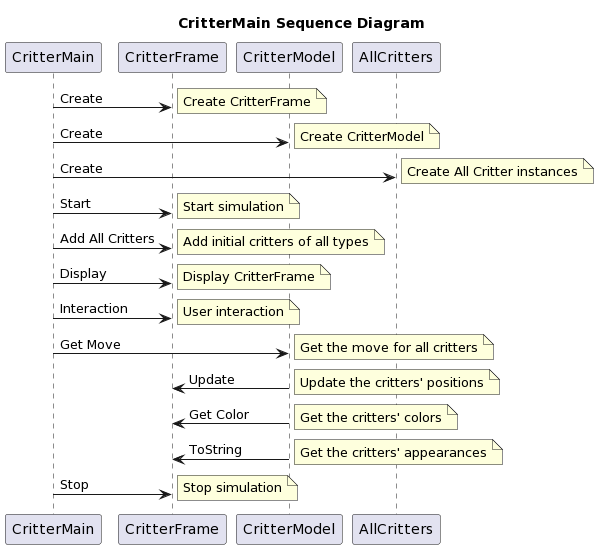
* 1. **NinjaCat Class**

|  |  |  |
| --- | --- | --- |
| **Purpose** |  | The NinjaCat class defines the behavior of the NinjaCat critter within the simulation. Its specific purpose is to exhibit unpredictable movements and teleport to a different location when infected. |
| **Method** |  | getColor(): Returns the color of the NinjaCat, which is pink.  toString(): Determines the visual representation of the NinjaCat, which is "NjC."  getMove(CritterInfo info): Defines the movement logic for the NinjaCat. NinjaCats move randomly in different directions, including hopping and infecting other critters. |
| **Control Structure** |  | The NinjaCat's control structure includes random movement logic, with a chance to teleport to a different location when infected. |
| **Input Requirements** |  | The NinjaCat class takes no input parameters for initialization. |
| **Output Results** |  | When executed, the NinjaCat exhibits unpredictable movements. It may even choose not to infect other critters even when it is facing it. |

1. **Program Workflows and Logic**
   1. **UML Class Diagram**

****

* 1. **UML Sequence Diagram**

****

1. **Development Details**
   1. **Development Environment**

The development of this project was carried out in a well-equipped software development environment. The following tools and technologies were used:

**Programming Languages**: Java was the primary programming language used for implementing the project.

**Integrated Development Environment (IDE):** The project was developed using IntelliJ IDEA, a powerful and user-friendly Java IDE. IntelliJ IDEA greatly facilitated coding, debugging, and project management.

* 1. **Project Structure**

The project was structured in a modular and organized manner to ensure maintainability and ease of development. The codebase was organized into the following main components:

**Critter Classes:** These classes encapsulate the behavior of different critters within the simulation. Each critter class (e.g., Bear, Tiger, NinjaCat) was organized into separate Java source files.

**User Interface:** The graphical user interface components (CritterFrame, CritterPanel) were separated from the core critter logic.

**CritterModel:** This core component managed the simulation grid, critter population, and simulation logic.

**CritterMain:** The entry point of the application that configured the simulation environment and initiated it.

* 1. **Coding Style and Standards**

The project adhered to Java coding conventions and style practices. Some key aspects of the coding standards included:

**Naming Conventions:** Descriptive and meaningful variable and method names were used to enhance code readability.

**Code Formatting:** Consistent code formatting, including indentation and code structure, was applied to maintain a clean and organized codebase.

**Comments and Documentation:** In-line comments were added to explain complex logic and provide information about the purpose of classes and methods. Javadoc-style comments were used to document classes and methods.

* 1. **Documentation**

Clear and comprehensive documentation was an integral part of the project:

**Inline Comments:** The source code was extensively documented with in-line comments to explain the logic of critical sections.

**Javadoc: Javadoc-**style comments were added to classes and methods, providing detailed descriptions and usage information.

1. **Results**
   1. **Testing and Debugging**

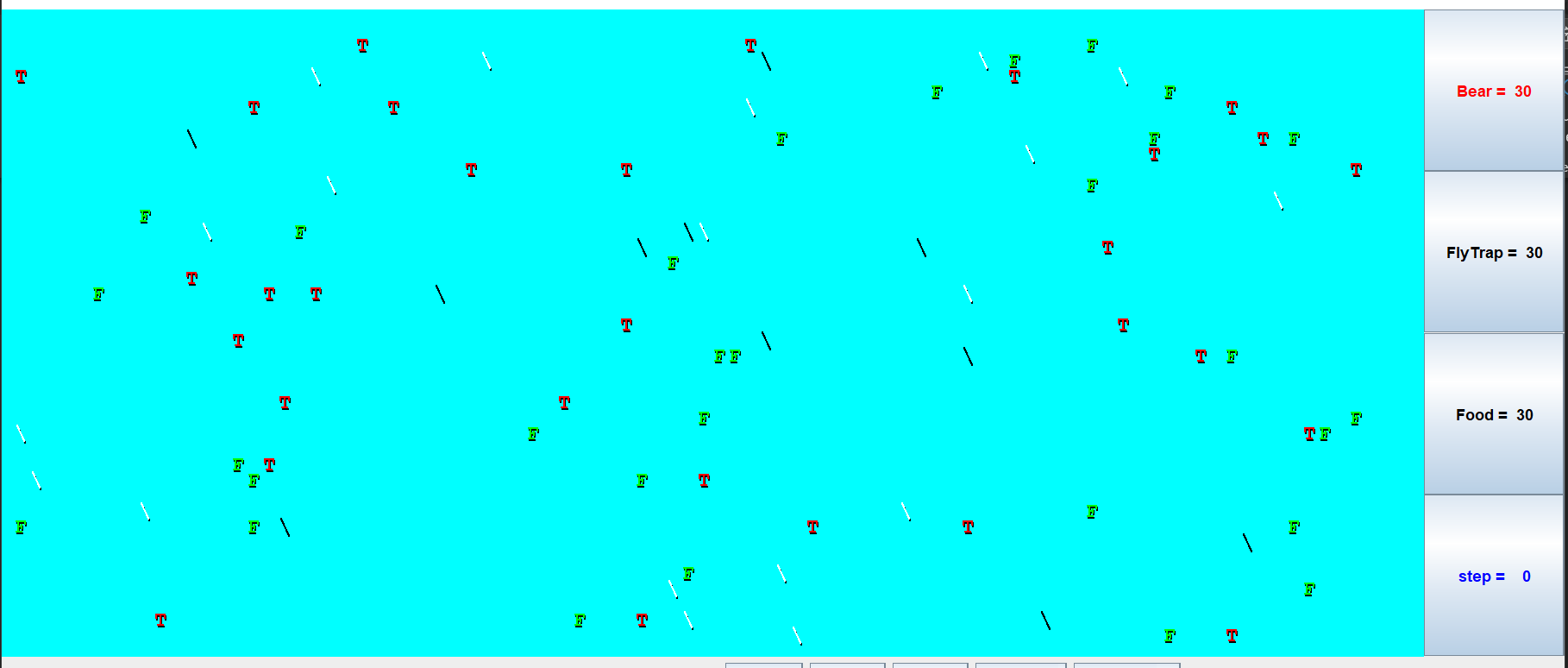
Extensive testing was conducted to ensure the correctness and reliability of the simulation:

**Unit Testing:** Individual critter classes were thoroughly tested to verify their behaviors and responses in different scenarios.

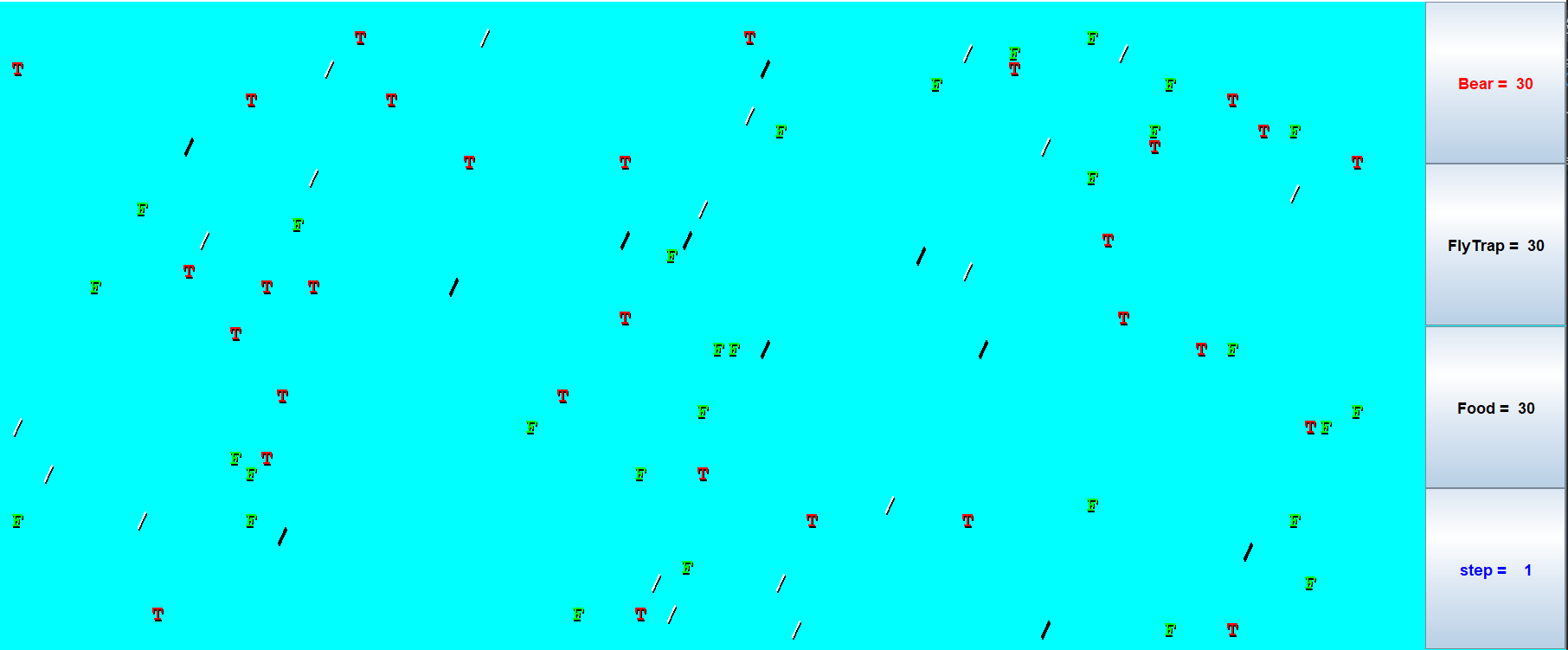
**Integration Testing:** The integration of critters within the simulation, as well as their interactions with the environment, was rigorously tested.

**Debugging:** Various debugging techniques were employed to identify and resolve issues, ensuring a stable and error-free simulation.

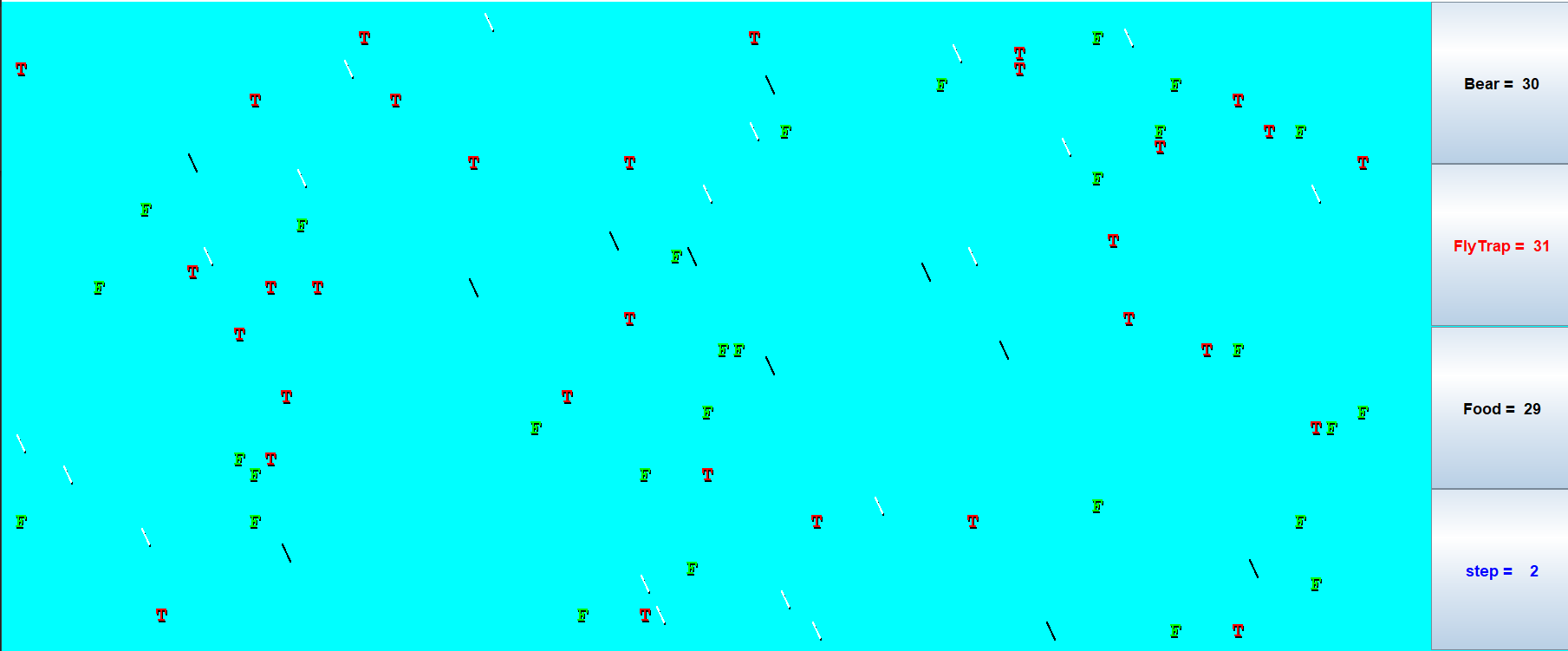
* 1. **Bear Testing and Results**

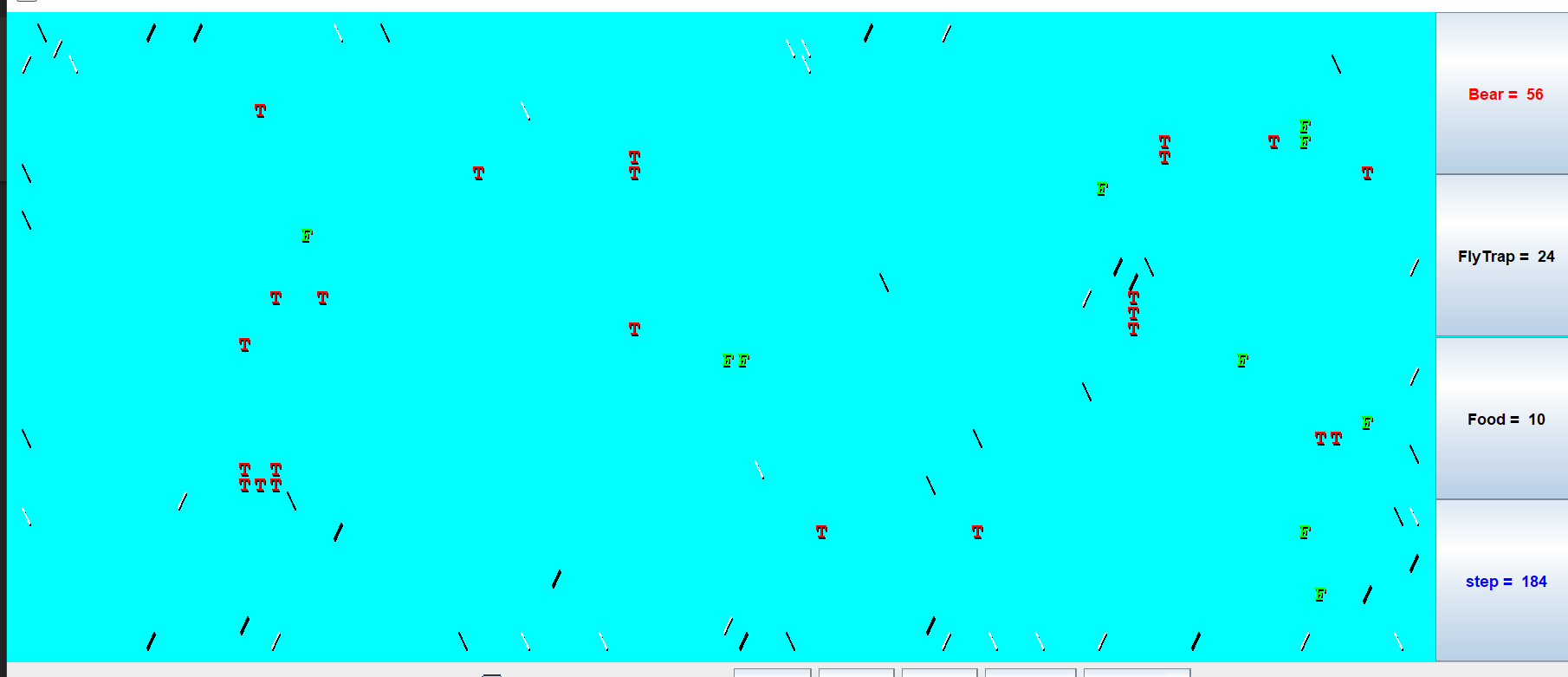
Run the simulator with just 30 bears in the world. Approximately half of them being white and about half being black. Initially all will be displayed with slash characters. 

When click "step", they should all switch to backslash characters.



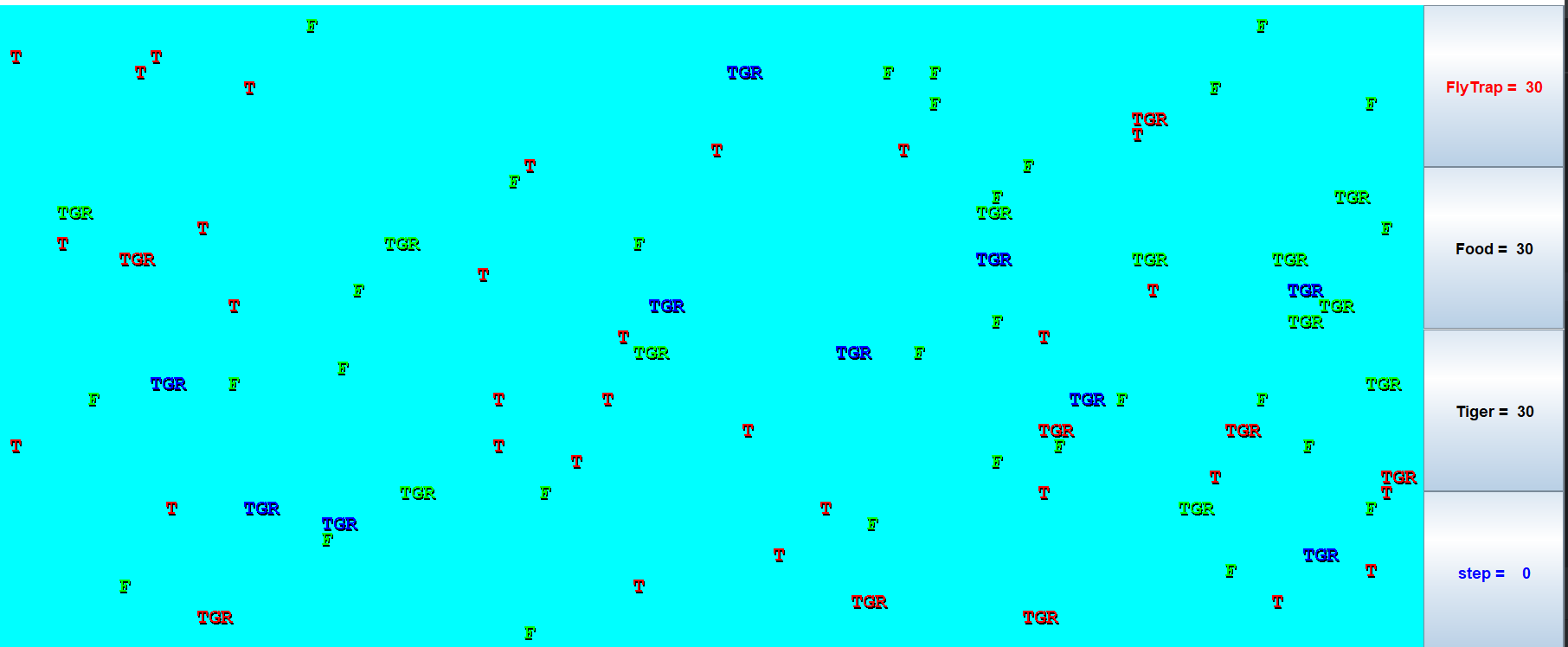
When you click "step" again they should go back to slash characters and so on.



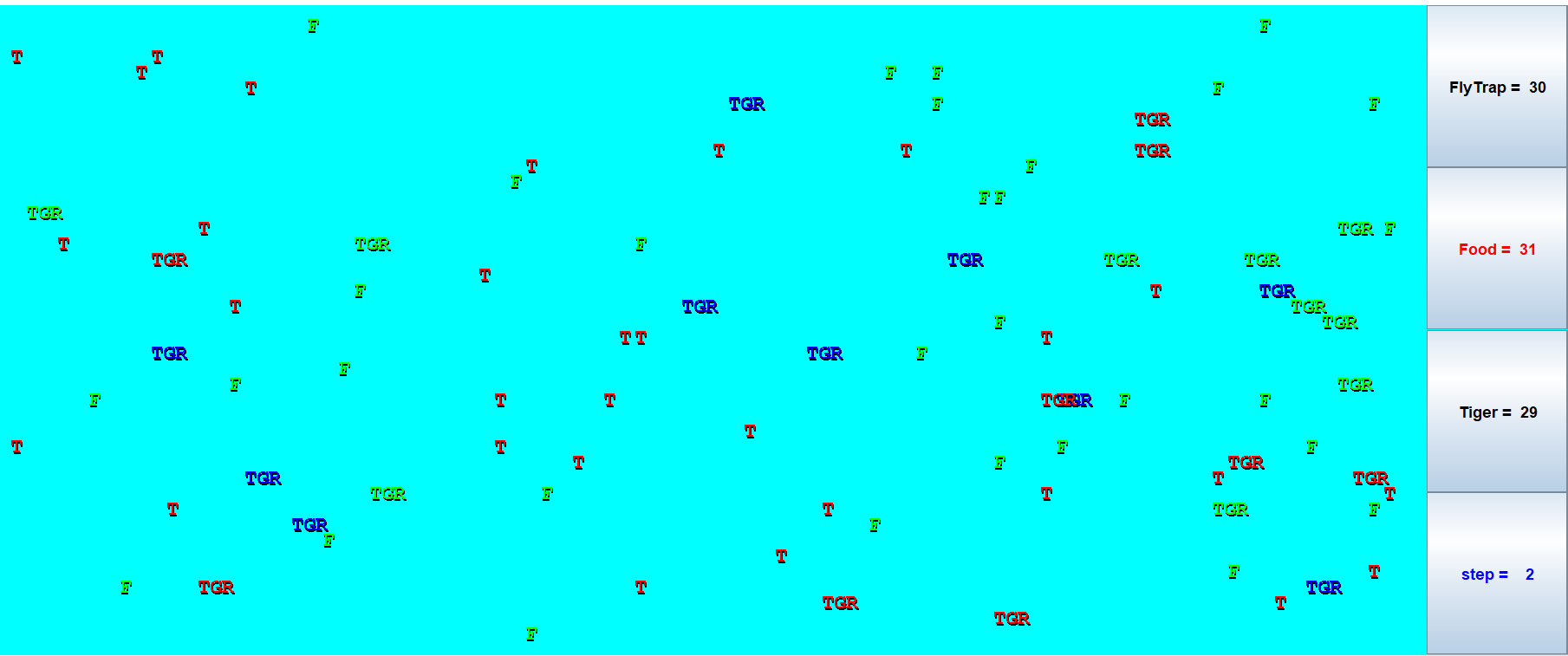
When you click "start", you should observe the bears heading towards walls and then hugging the walls in a counterclockwise direction. They will sometimes bump into each other and go off in other directions, but their tendency should be to follow along the walls. 

* 1. **Tiger and WhiteTiger Testing and Results**

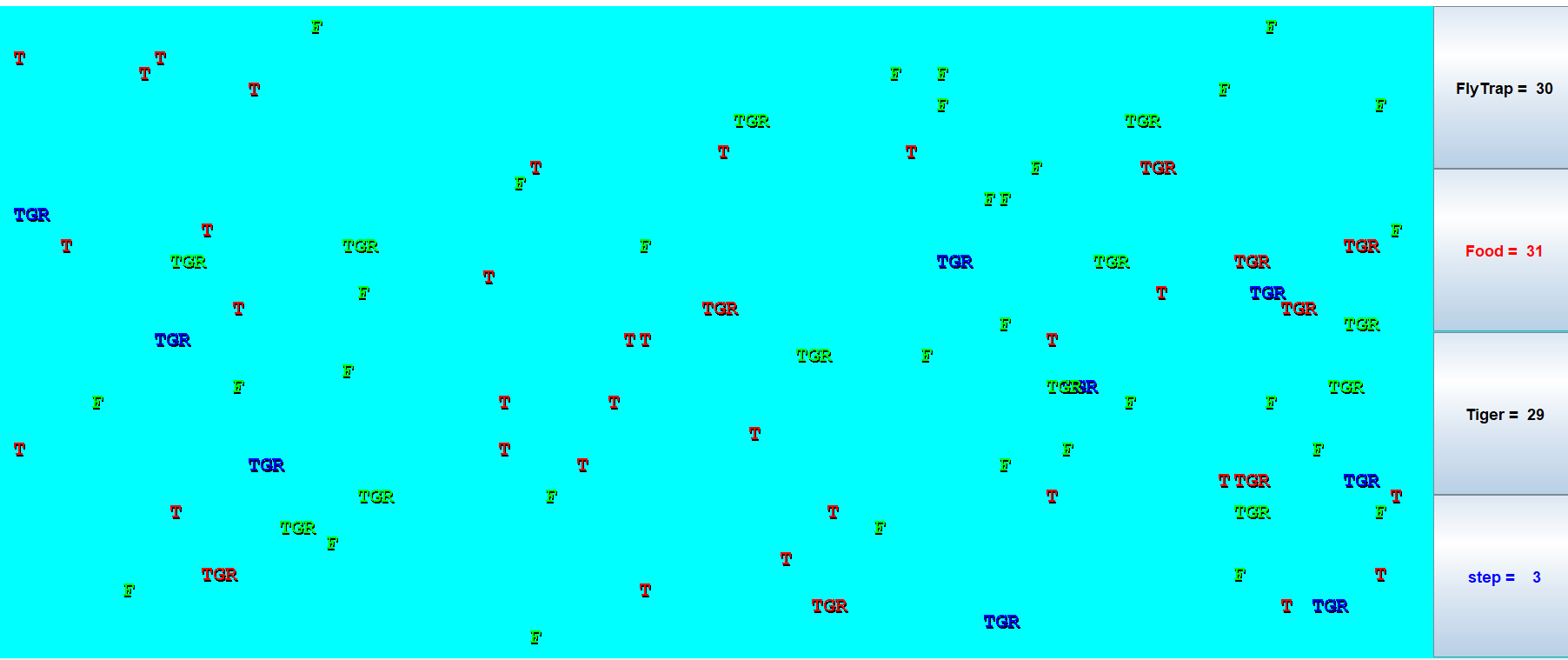
Run the simulator with just 30 Tigers in the world. About one third of them being red and one third being green and one third being blue.

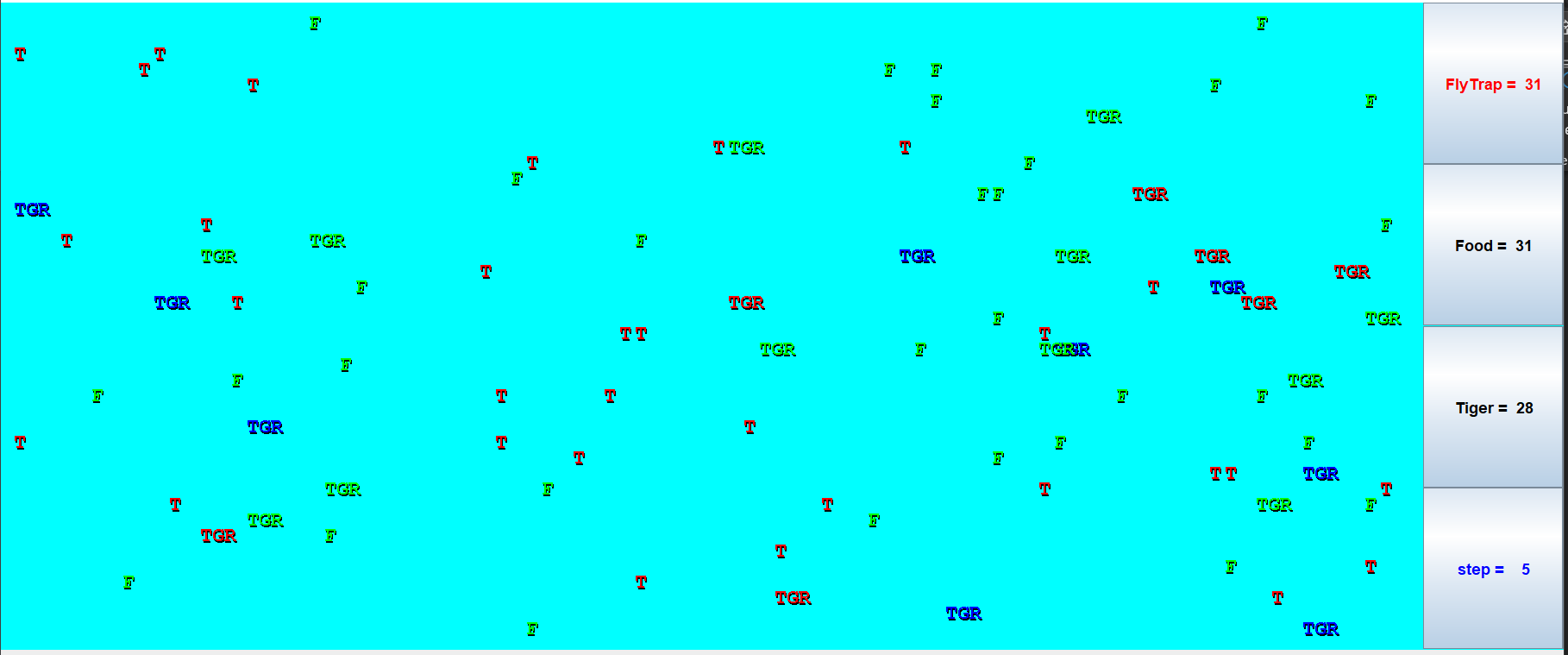


Use the "step" button to make sure that the colors alternate properly. They should keep these initial colors for three moves. That means that they should stay this color while the simulator is indicating that it is step 0, step 1, and step 2.

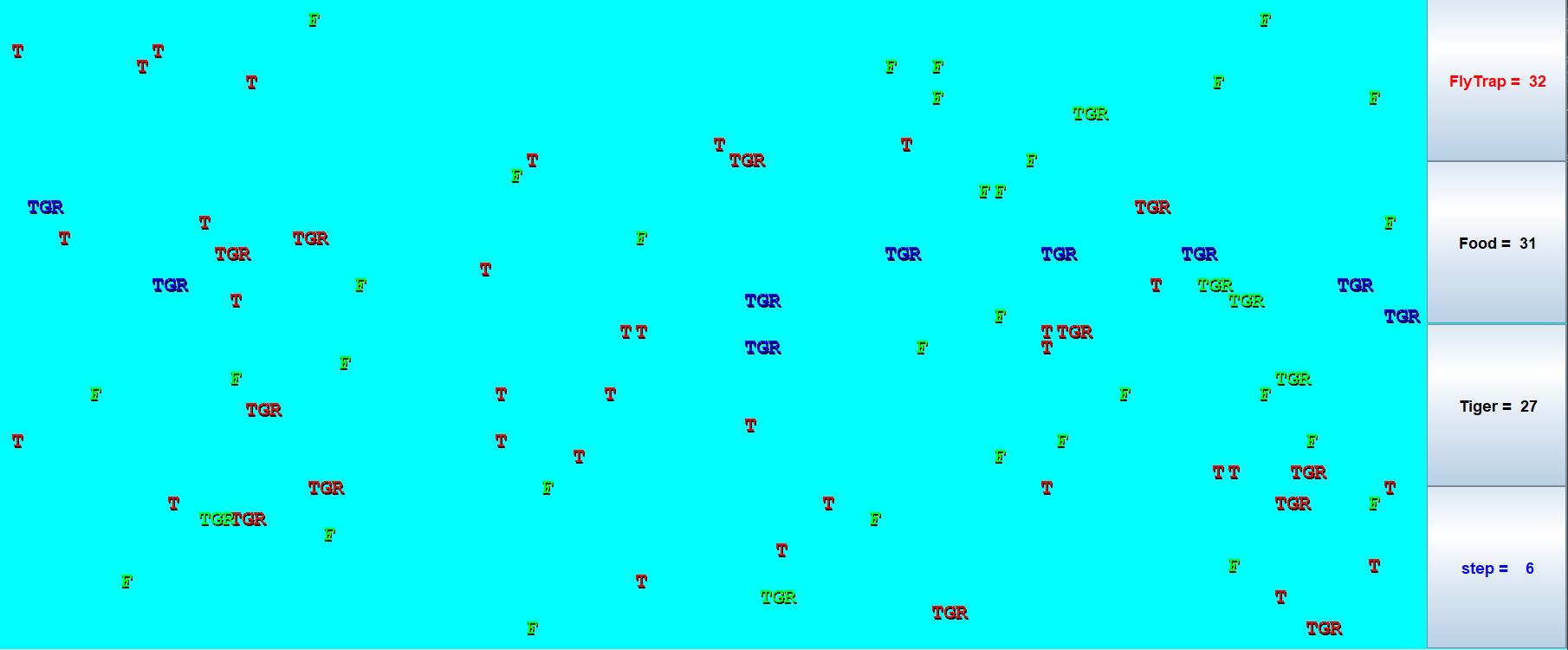


They should switch colors when the simulator indicates that you are up to step 3 and should stay with these new colors for steps 4 and 5.

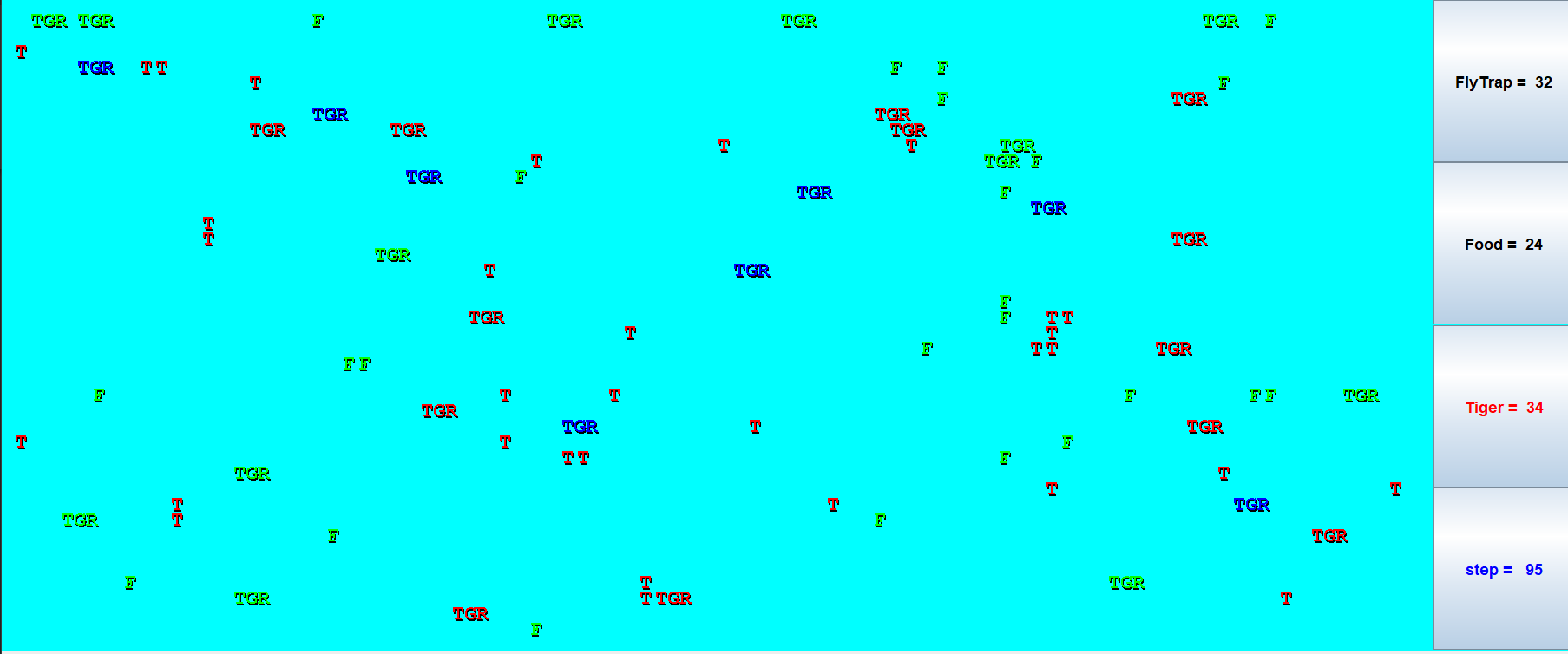




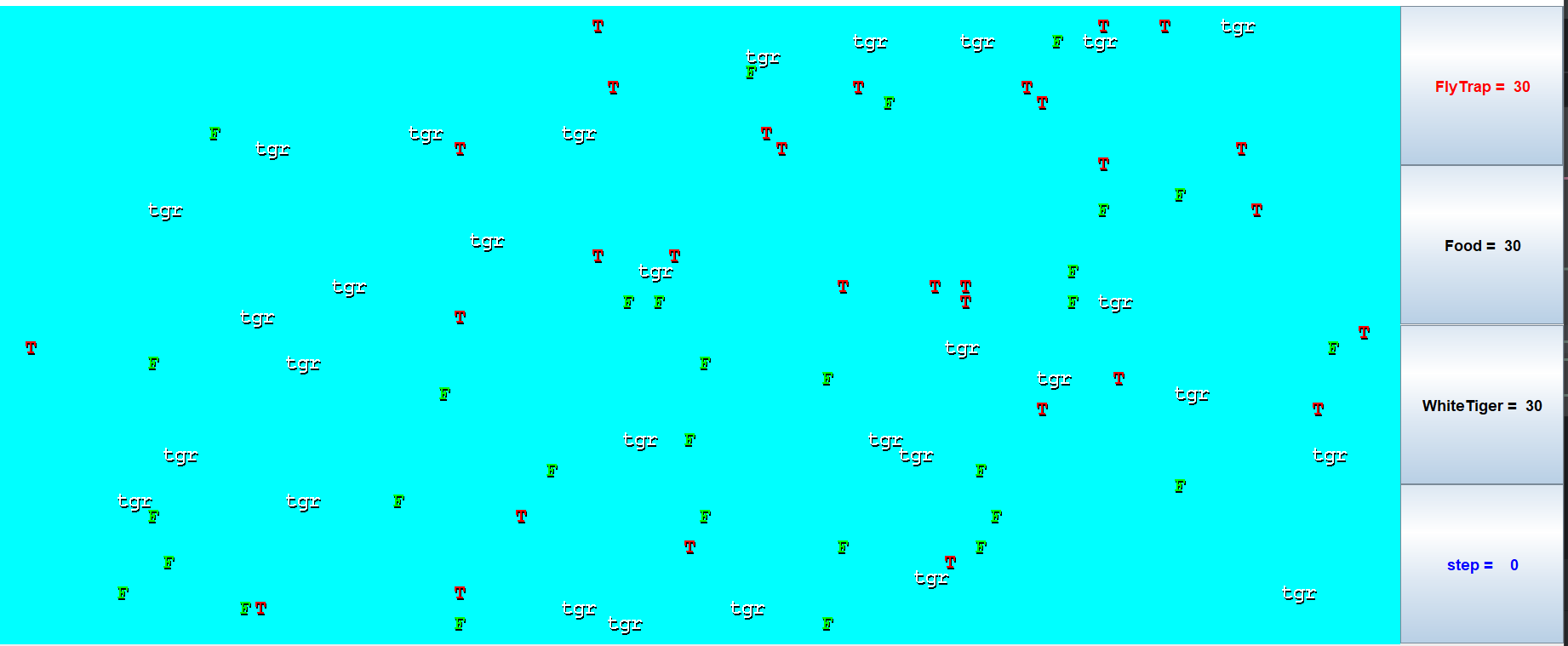
Then you should see a new color scheme for steps 6, 7, and 8 and so on.

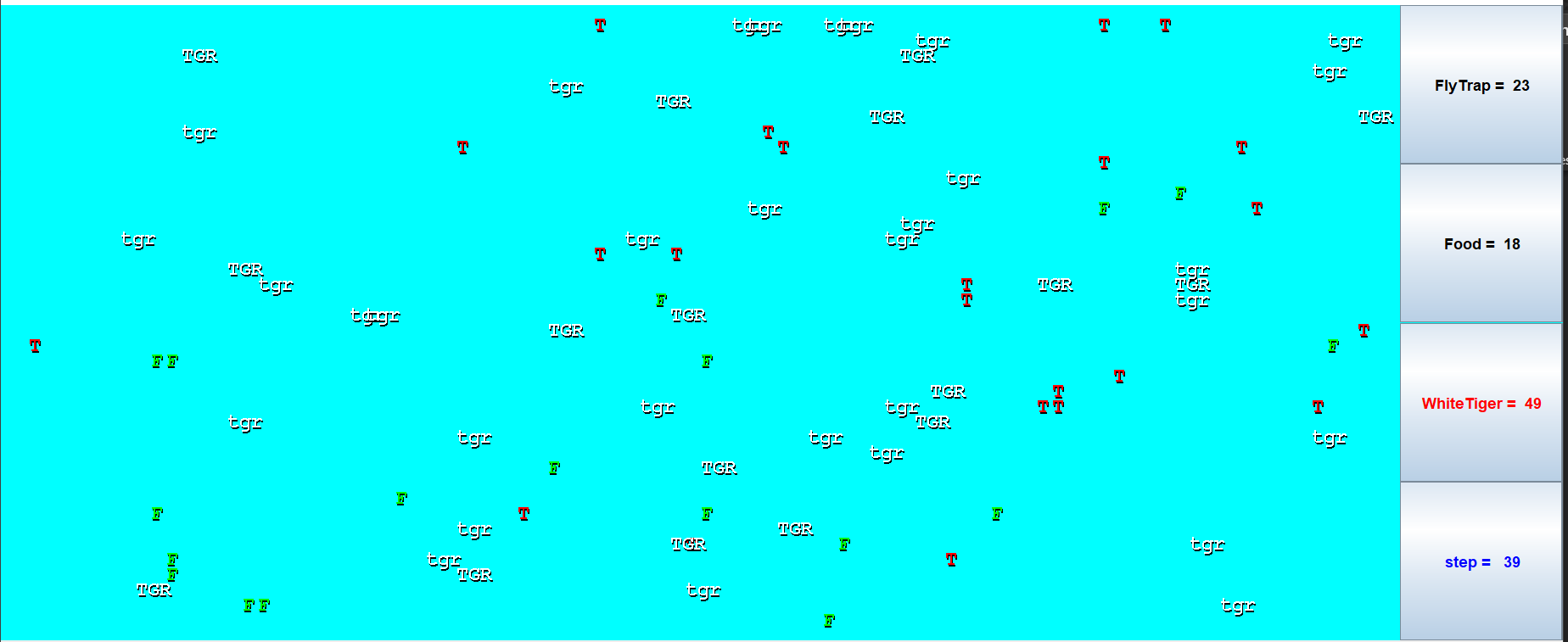


When you click "start" you should see them bouncing off of walls. When they bump into a wall, they should turn around and head back in the direction they came. They will sometimes bump into each other as well. They shouldn’t end up clustering together anywhere.



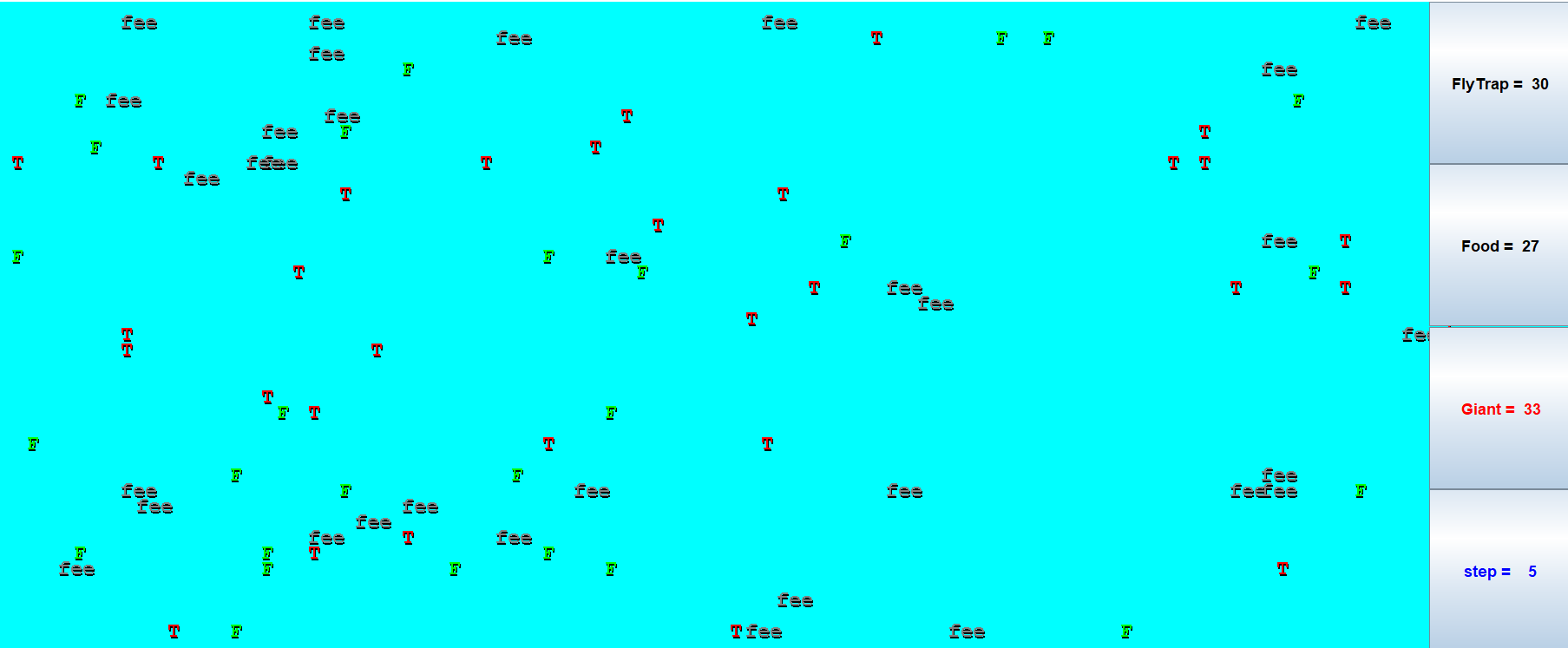
WhiteTiger: This should behave just like a Tiger except that they will be White. They will also be lower-case until they infect another Critter, then they "grow up".



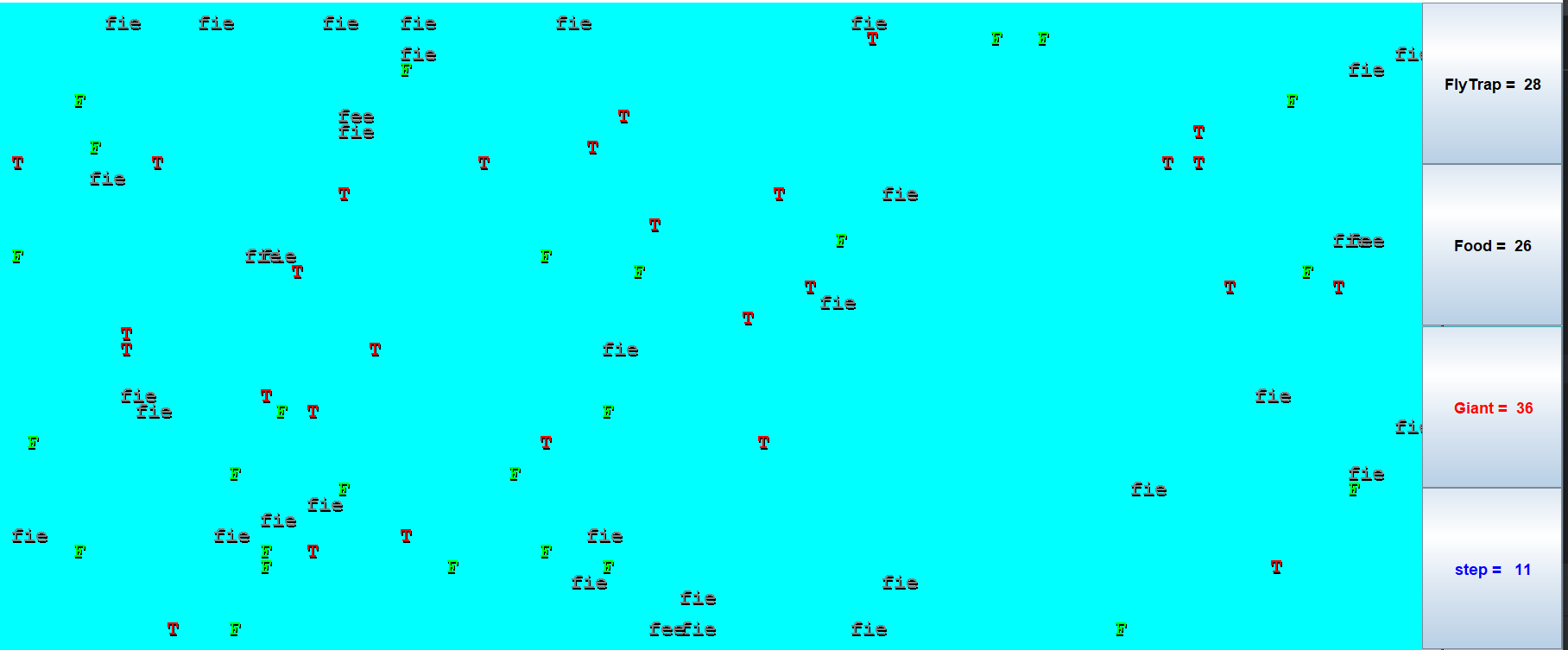


* 1. **Giant Testing and Results**

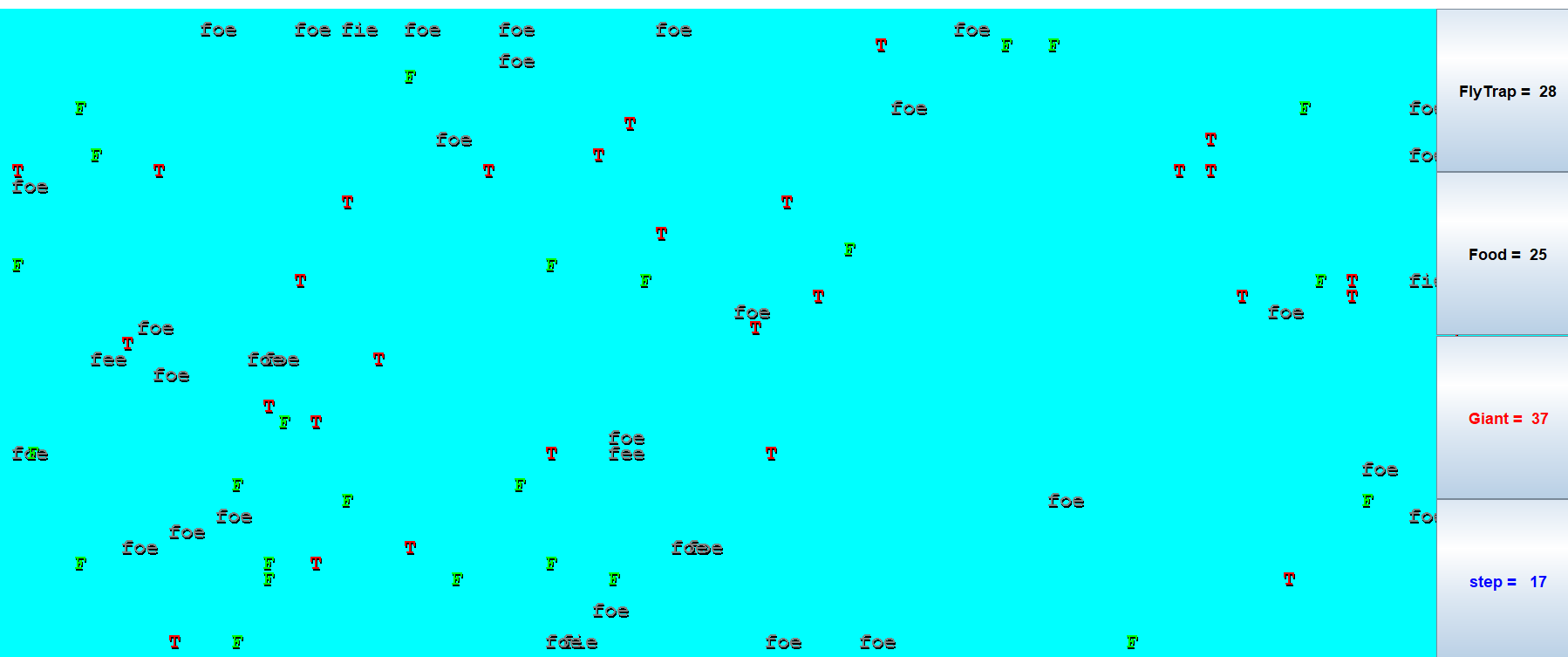
Run the simulator with just 30 giants in the world. They should all be displayed as "fee". This should be true for steps 0, 1, 2, 3, 4, and 5.



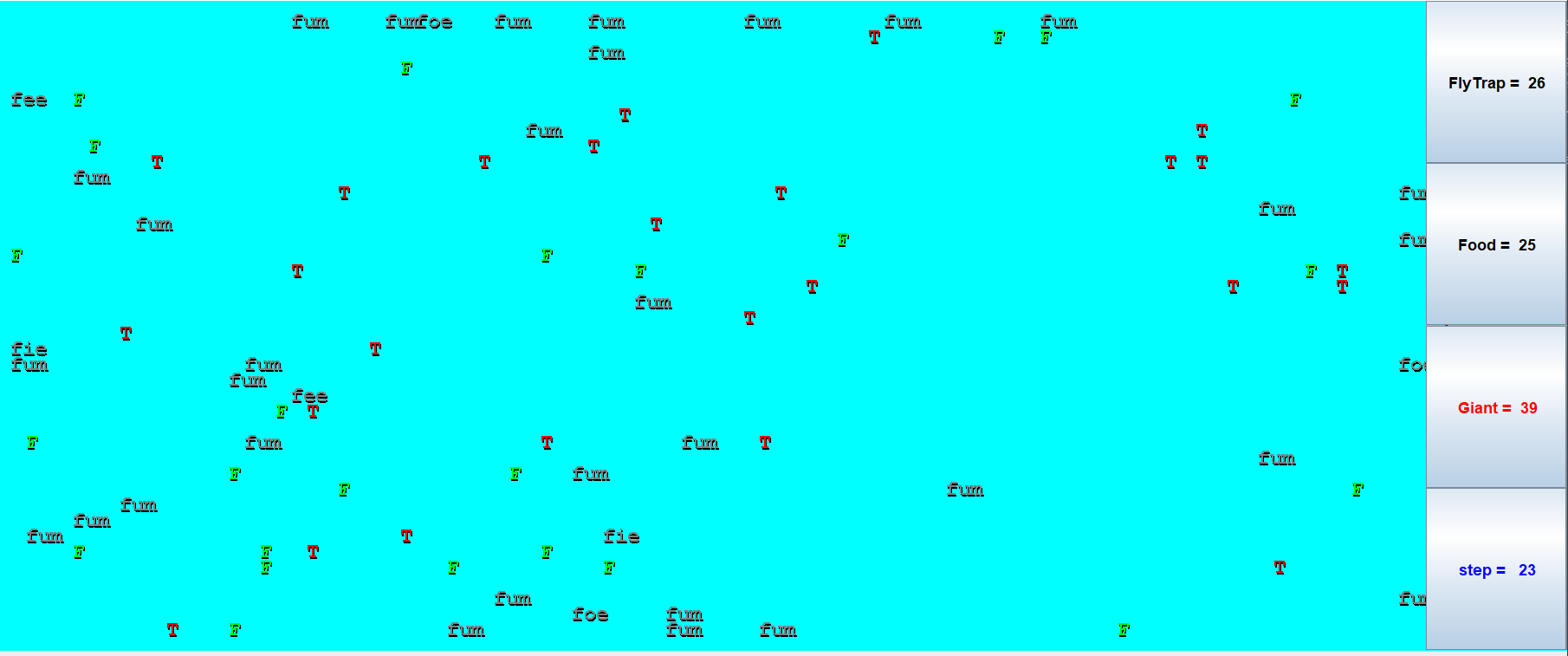
When you get to step 6, they should all switch to displaying "fie" and should stay that way for steps 6, 7, 8, 9, 10, and 11.



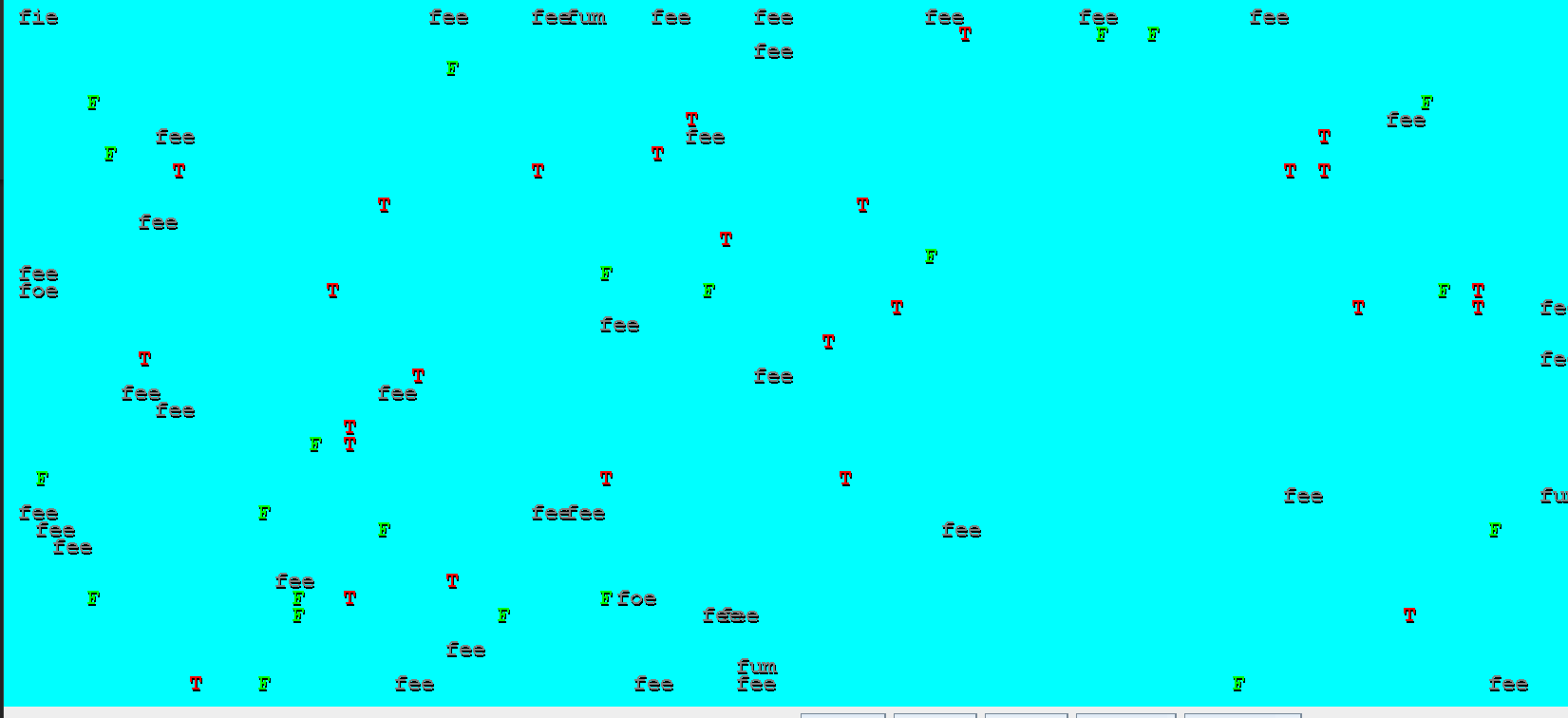
Then they should be "foe" for steps 12, 13, 14, 15, 16, and 17.



And they should be "fum" for steps 18, 19, 20, 21, 22, and 23.



Then they should go back to "fee" for 6 more steps, and so on.



1. **Discussion**

The "Discussion" section provides an opportunity to delve into the various aspects of the project and analyze its implications, challenges, and outcomes.

* 1. **Critter Behaviors and Interactions**

One of the core objectives of this project was to simulate the behavior of diverse critters within an enclosed environment. The critter classes, including Bears, Tigers, NinjaCats, and others, were designed with unique behaviors. During the development process, we encountered challenges in defining and implementing these behaviors to create a dynamic and engaging simulation.

The interactions between critters, including predation, competition, and cooperation, were central to the simulation's realism. Our discussion revolves around the success of these interactions, the accuracy of the critters' behaviors, and the balance between different critter types. For instance, Bears infected other critters, while Tiger critters occasionally changed color, reflecting their simulated emotions.

* 1. **Future Improvements**

While the project meets its current objectives, there are opportunities for future enhancements, including:

**Additional Critters:** Expanding the variety of critter types to make the simulation more diverse and engaging.8.2

**Interactive Features:** Incorporating interactive elements and user controls to allow users to influence the simulation.

**Real-time Configurability:** Enabling real-time adjustments to the simulation parameters, such as critter population and environmental conditions.

1. **Final Remarks and Conclusion**

I would like to express my gratitude to the developers and contributors of the original Critter project, upon which this project is based. Their work has served as an invaluable source of inspiration and guidance throughout the development process.

In conclusion, this project stands as a testament to the potential of software as an educational tool. It successfully combines the worlds of computer science and ecology to offer an interactive learning experience. The project's future holds exciting possibilities for further development and expansion, ensuring that its impact continues to grow in the field of education and beyond.