**Extension 3: (Grace)**

- **let agents die when they are unhappy and not able to be happy after a certain of moves -> they move out of the city。**

According to extension 3, I have developed a more specific scenario based on the keywords "Sweden," "student apartment," and "cost of living."

The basic criteria I have defined are as follows: the destination is Linköping, Sweden; the target is a student apartment. The desired attributes include rent between 4000 SEK and 7000 SEK, an apartment size of around 25 square meters, and amenities such as a kitchen and bathroom. The individual's potential preference is to be closer to the school. When making migration decisions, individuals consider whether the rent level is within their affordability range and also factors such as apartment size and proximity to the school. Based on these factors, each potential student apartment is assigned a score. Individuals can choose the most suitable destination based on the scores, with a score of 0 meaning the individual is completely dissatisfied and no matching student apartment is available. Scores between 2 and 3 mean they will continue searching for a suitable apartment, while scores between 4 and 5 indicate complete satisfaction, and the turtle stops moving.

# 1.globals

# 2.turtles-own:

- **happy?**: Indicates whether each turtle is happy or not. It represents whether at least the target percentage of its neighbors have the same color as the turtle.

- **similar-nearby**: The number of neighboring patches that have the same color as the turtle.
- **total-nearby**: The total number of neighboring patches.
- **migration-preference**: The individual preference value for migration destination.
- **memory**: The memory attribute used to store the previous migration decisions.
- **apartment-score**: The score assigned to apartments based on individual preferences.

# 3.patches-own

The patches-own block defines the properties of patches in the model. Here are the descriptions of these patch properties in English:

- **rent-level**: Represents the rental level of the patch.
- **apartment-size**: Represents the size of the apartment on the patch.
- **distance-to-school**: Represents the distance from the patch to the school.

These patch properties provide information about the characteristics of each patch in the model, such as the rental level, apartment size, and distance to the school.

# 4.setup

he **setup** procedure is used to initialize the settings of the model. It performs the following steps:

1. Clears the entire model, removing all turtles and patches.
2. Generates turtles on a random selection of patches. The number of turtles is determined by the **number** variable.

3. Sets the color and migration preference value for each turtle. The color is randomly assigned as either blue or yellow, representing different groups. The migration preference value is a random float between 0.0 and 1.0, indicating the individual's preference for migration destination.
4. Creates student apartments by calling the **create-apartments** procedure.
5. Updates the turtles and global variables by calling the **update-turtles** and **update-globals** procedures.
6. Resets the model's tick counter to 0.

This setup procedure prepares the initial state of the model, placing turtles on patches, assigning their attributes, creating apartments, and updating relevant variables to start the simulation.

# 5. create-apartments

In the **create-apartments** procedure, a list called **potential-apartments** is defined to store potential student apartments. Each student apartment has three attributes: rent level, apartment size, and distance to school.

Then, using an **ask** block with turtles, the following operations are performed for each turtle:

1. Variables such as maximum rent, minimum rent, maximum apartment size, minimum apartment size, maximum distance to school, and minimum distance to school are defined.
2. A **while** loop is used to iterate through each potential student apartment in the **potential-apartments** list.
3. The rent level, apartment size, and distance to school of the current potential student apartment are obtained.

4. The apartment score (**apt-score**) is calculated based on the individual's preference and the attributes of the potential student apartment.
5. The **apt-score** is composed of the rent level, apartment size, and distance to school, and each factor's contribution to the score is measured using normalized values.
6. The current student apartment in the **potential-apartments** list is updated, setting its score attribute to the computed **apt-score**.

Through this procedure, the score is calculated for each potential student apartment, and the scores are stored in the corresponding entry in the **potential-apartments** list.

# 6.to-report

# 7.to-go

The **go** procedure is the main loop of the model. It first checks the stopping condition of the model, which is to stop if all turtles are happy or if the specified number of time steps (100 ticks) has passed. Then, it sequentially calls the **move-unhappy-turtles**, **update-turtles**, and **update-globals** procedures, and increments the time step (tick).

# 8. to move-unhappy-turtles

The **move-unhappy-turtles** procedure is responsible for handling unhappy turtles. It first finds a new spot (using the **find-new-spot** procedure), then evaluates potential apartments (using the **evaluate-potential-apartments** procedure), selects the best apartment (using the **select-best-apartment** procedure), and if there is a selected apartment, it moves the turtle to that apartment and updates the turtle's memory.

## 9.to find-new-spot

The **find-new-spot** procedure is used to find a new spot for the turtle in the model. It first randomly selects an angle and a distance to move. Then, it checks if there are any other turtles at that location. If there are other turtles, it recursively calls the **find-new-spot** procedure until it finds a location without any other turtles. Finally, it moves the turtle to the center of that location.

## 10. to evaluate-potential-apartments

The **evaluate-potential-apartments** procedure is used to evaluate potential apartments. It first selects potential apartments that meet specific criteria (rent-level, apartment-size, and distance-to-school within certain ranges). Then, it sets the color of these potential apartments to white, calculates a score based on individual preferences and the attributes of the potential apartments, and displays the score on the apartment patch.

## 11.to select-best-apartment

The **select-best-apartment** procedure is used to select the highest-rated apartment from the potential white apartments as the best apartment. It first selects the white potential apartments, then finds the apartment with the highest score, and returns that apartment. If there are no potential apartments, it returns **nobody**.

## 12.update-turtles

The **update-turtles** procedure is used to update the state of turtles. For each turtle, it calculates the number of neighboring turtles that have the same color as itself (**similar-nearby**), counts the total number of neighbors (**total-nearby**), and determines whether the turtle is happy based on the minimum desired similarity ratio (**%-similar-wanted**).

## Conclusion

By extending this model, we can simulate and study the process of a population with migration behavior in search of satisfactory living conditions. This extension allows us to consider more factors, such as rent levels, apartment sizes, and distances to schools, to more accurately reflect individuals' preferences for living conditions.

The significance of this extension lies in providing a tool to study population migration decisions. By simulating the migration behavior of turtles, we can observe how different individuals' preferences influence their decisions in choosing residential locations and investigate the impact of different factors on individual satisfaction and overall community structure.

Through the extension of the model, we can delve deeper into studying the influence of different factors on migration behavior and community formation. By adjusting parameters and conditions, we can simulate migration decisions under different scenarios, such as different rent levels, apartment sizes, and distance ranges to schools. This can help us better understand the decision-making process of populations when choosing residential locations, providing valuable insights for urban planning and community development.

Furthermore, by observing the results of the model's execution, we can evaluate the stability and satisfaction levels of communities under different conditions. We can test the impact of different strategies on community stability by adjusting satisfaction thresholds and migration decision rules, providing guidance for community planning and policy-making.

In summary, through the extension of this model, we can gain a deeper understanding of the dynamics of population migration and community formation, and generate valuable insights for

urban planning, community development, and policy interventions related to segregation and residential choices.