KOÇ ÜNİVERSİTESİ
COMPUTER ENGINEERING

COMP132: Advanced Programming

Programming Project Report

Food Chain Through Time: Simulation Design and Development

<Duygu Yunus, 89824>

<Fall 2025>

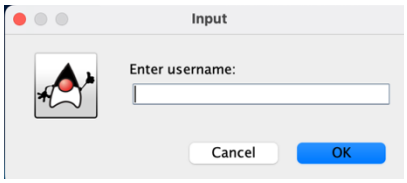
Part 1

Logging Into the Game

1. Username Selection

The player enters a username, which is displayed in the game HUD and saved to the game state.

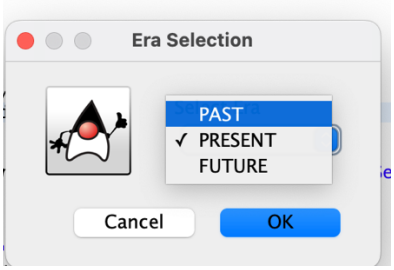
Figure 1.



2. Era Selection

The user selects of the following eras: Past, Present, Future. Each era affects: Background image, animal photos and food.

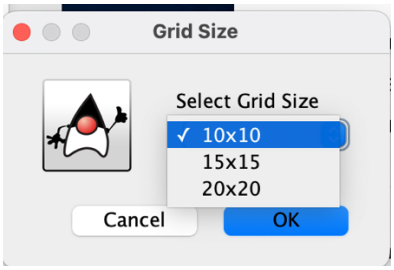
Figure 2.



3. Grid Size Selection

The user chooses between 10x10, 15x15, and 20x20.

Figure 3.

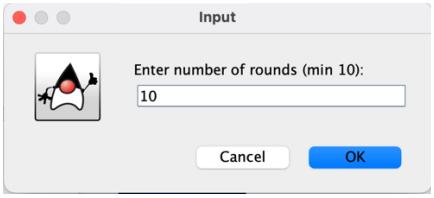


4. Number of Rounds Selection

The player enters the maximum number of rounds (minimum of 10).

The game ends automatically once this limit is reached.

Figure 4.



5. Food Chain File and Era Differences

Each era loads its initial setup from a corresponding text file using the FoodChainLoader class.

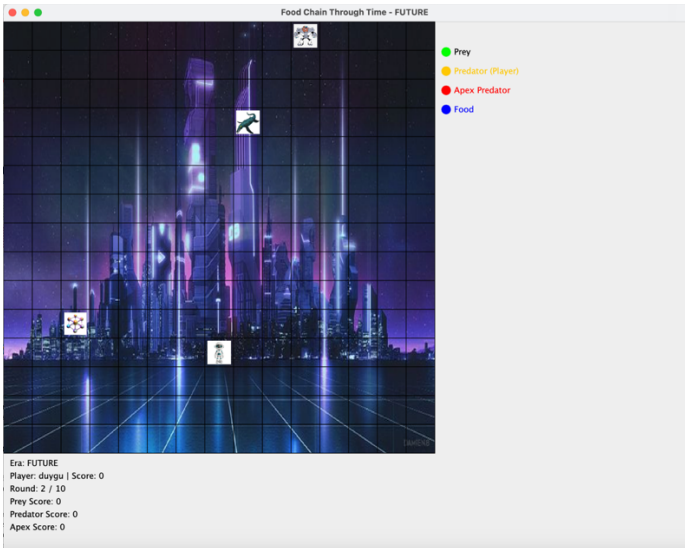
Changing the food chain text file directly changes:

- Which animals are spawned
- The number of food items
- Initial positioning behaviour

This demonstrates that the game logic dynamically adapts to external configuration files without requiring code changes.

6. Gameplay Mechanics

Figure 5.



7. Player Movement

- The player controls the Predator using keyboard arrow keys.
- Each movement advances the game by one round.
- The grid visually updates after every move.

8. Special Moves

Special moves are implemented using a cooldown-based system:

8.1 Predator (Player)

Can perform a special dash move. Activated using **SPACE** or **SHIFT**. Cooldown: 2 rounds.

```
@Override
public void useSpecial(GameState state) {
    if (!canUseSpecial()) return;
    position.setX(position.getX() + 2);
    specialCooldown = 2;
}
```

8.2 Apex Predator

Uses AI-based chase behaviour to pursue the nearest target**.** A special ability exists in the codebase, but movement is primarily driven by AI logic

9. AI Behaviour

9.1 Prey AI

- Prey evaluates nearby positions and chooses the safest move.
- It prioritizes:
    - Distance from predators
    - Proximity to food
- Positions near predators are heavily penalized.

9.2 Apex Predator AI

- Apex Predator actively hunts the closest Prey or Predator.
- Every second round, it moves strategically toward the nearest target.
- This creates increased pressure on both AI and player.

10. Collisions and Scoring

10.1 Food Consumption

- When a Prey moves onto food:
    - +3 points are awarded
    - The food respawns at a random location

10.2 Animal Interactions

- Predator eats Prey → Predator +3, Prey −1
- Apex eats Prey or Predator → Apex +1, Victim −1
- Consumed animals respawn at a random empty cell

11. Respawn Behaviour

Prey and Predator animals respawn at random valid grid positions. Food items respawn at random locations after being consumed. Respawning never overlaps with another animal.

12. Game Completion

The game automatically ends once the maximum round count is reached. The winner is determined based on the highest score. If multiple roles share the highest score, the game declares a tie and displays multiple winners.

Figure 6.

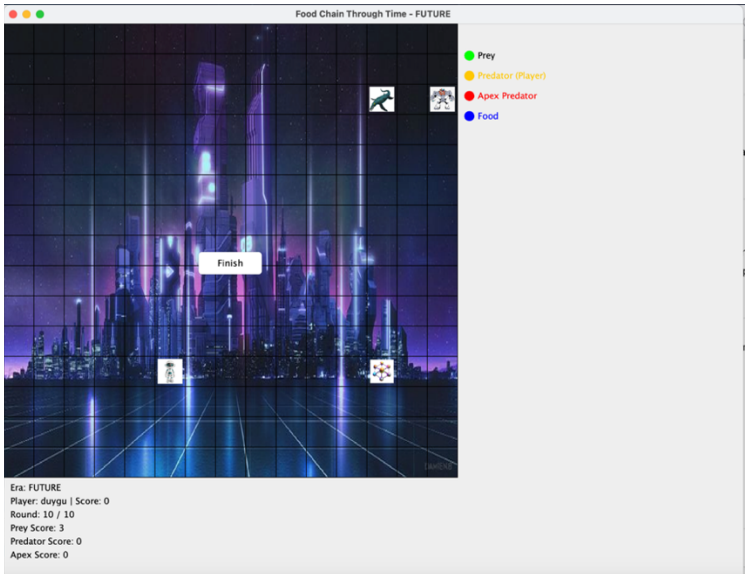
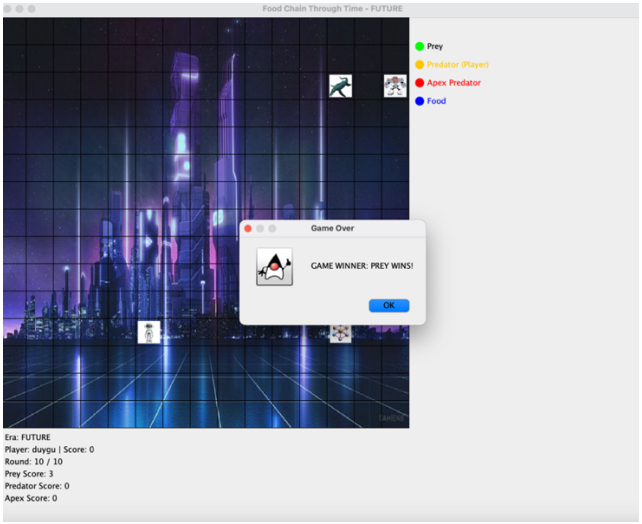


Figure 7.

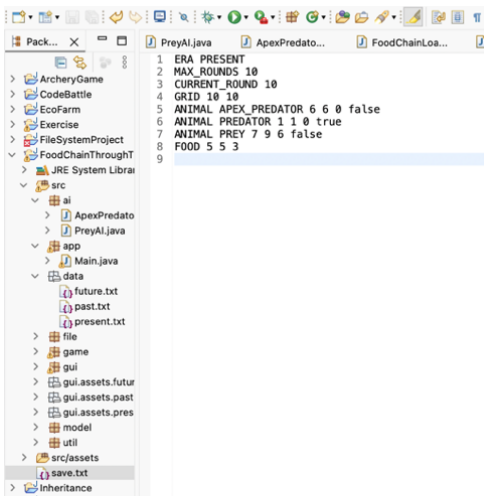


13. Saving and Loading the Game

13.1 Saving

- At the end of the game, the full game state is saved to a text file.
- The saved data includes: era, grid size, current round, animal roles, animal positions, animal scores, food positions and food values.

Figure 8.



13.2 Loading

- A previously saved game state can be loaded from the text file using the game's loading functionality.

Logs

- Each round start is logged to a log file using GameLogger.
- Logs provide a chronological record of gameplay progression.

Part 2

1. Package Characters

1.1 Animal Class

The Animal class represents the most fundamental entity in the game. It is declared as an abstract class and provides shared attributes and behaviour for all animal types. Store position, role, and score information. Handle cooldown logic for special abilities. Define abstract methods to be implemented by subclasses.

Key Attributes

- Position position – current location on the grid
- Role role – role type (Prey, Predator, Apex Predator)
- int score – accumulated points
- int specialCooldown – cooldown counter for special abilities

Core Methods

- getPosition() / setPosition() – Access and update position
- increaseScore(int amount) – Modifies the animal's score
- canUseSpecial() – Determines whether the special ability can be used
- tickCooldown() – Updates the cooldown state every round

Abstract Methods

- move(Position newPosition) – Defines how movement is applied
- isPlayerControlled() – Distinguishes player-controlled and AI-controlled roles
- useSpecial(GameState state) – Role-specific special behaviour

1.2 Position Class

It is a simple data class used to represent coordinates on the grid. It stores x and y coordinates. Provide getter and setter methods.

1.3 Prey Class

Extends Animal and represents a non-player-controlled role. Controlled entirely by ai logic. Cannot use special abilities. Focuses on survival and food collection.

Key Behaviours

- Implements move() for position updates
- Overrides isPlayerControlled() to return false
- Overrides canUseSpecial() to return false

1.4 Predator Class

The Predator class represents the player-controlled character and extends Animal. Controlled by keyboard input. Supports a special dash moveStores the player's username. Moves two grid cells horizontally.

Key Overrides

- isPlayerControlled() returns true
- useSpecial(GameState state) implements dash logic

This separation allows the Predator to behave differently from AI-controlled roles while still sharing common logic through inheritance.

1.5 ApexPredatorClass

The ApexPredator class extends Animal and represents the strongest AI-controlled role. Fully AI-controlled Uses chase-based movement logic. Has a special ability defined, though primary movement is driven by AI.

Key Overrides

- isPlayerControlled() returns false
- useSpecial(GameState state) defines a longer dash with a cooldown of 3 rounds

Movement decisions for the Apex Predator are handled centrally in the game controller to maintain consistent AI behaviour.

1.6 Food Class

The Food class represents consumable items placed on the grid.
Attributes

- Position position – Location of the food item
- int value – Score value awarded when consumed

Food objects interact with animals through collision detection handled by the controller.

2 Package Helpers

2.1 Role Enumeration

The Role enum defines the three distinct roles in the game:

- PREY
- PREDATOR
- APEX_PREDATOR

Using an enumeration ensures type safety and prevents invalid role assignments.

2.1 Era Enumeration

The Era enum represents the temporal setting of the game and is used to control visual and content-related differences.,

Defined Eras

- PAST
- PRESENT
- FUTURE

Each era is associated with: A folder name for asset organization, background image path.

public String getSpritePath(String name) Method: Returns the file path of role-specific sprites for the selected era.

3. Package Gui

3.1 GameFrame Class

Represents the main application window of the game. It initializes the game interface based on the selected grid size, sets the window title using the chosen era. It also handles basic window configuration and allows the game state to be updated when a new state is loaded.

public void setGameState(GameState newState) Method: It is used to update the game when a new game state is loaded. The parameter newStaterepresents the newly loaded or updated game data, including the grid, animals, food, and round information. First, the variable state is updated to reference this new game state. Then, new GameController object is created and stored in the variable controller so that all game logic operates on the updated state. After that, panel.setState(newState) updates the GamePanel to display the new game data. Finally, repaint() is called to refresh the window and visually reflect all changes on the screen. This method ensures that both the game logic and the user interface remain synchronized after loading a new game state.

public GameFrame(GameState state) Method: The GameFrame constructor creates and sets up the main game window. The parameter state represents the current game data and is stored so the frame can access it. A GameController is created to manage the game logic. The size of the window is calculated using the grid size and a fixed cell size. The window title is set based on the selected era, and basic window settings such as closing behaviour and screen position are configured. Finally, a GamePanel is added to the frame, the window is displayed, and keyboard focus is given to the game panel so the player can control the game.

3.2 GamePanel Class

It displays the game on the screen and handles the player interaction. It draws the game grid, animlsi food, background. it listens to keyboard input to control the player. It also updates the game after each move and shows the game-over screen when the game ends.

public GamePanel(GameState state) Method: The constructor initializes the game panel using the current game state. It creates a GameController to manage game logic, loads era-specific images, and enables keyboard input. A key listener is added to detect player actions such as movement and special abilities. The constructor also creates a “Finish” button, which becomes visible when the game ends.

protected void paintComponent(Graphics g) Method: It draws all visual elements of the game. It creates a GameController to manage game logic, loads era-specific images, and enables keyboard input. A key listener is added to detect player actions such as movement and special abilities. The constructor also creates a “Finish” button, which becomes visible when the game ends and allows the user to close the application.

private void drawGrid(Graphics g) Method: It visually separates the game board into individual cells.

private void drawFood(Graphics g) Method: Draws all food items on the grid. Each food object’s position is converted into screen coordinates, and the corresponding food image is displayed inside the grid cell.

private void drawAnimals(Graphics g) Method: Draws all animals on the grid. The displayed image depends on the role of each animal (Prey, Predator, or Apex Predator). This visual distinction makes it easy to identify different roles during gameplay.

private void drawHUD(Graphics g) Method: Displays game information below the grid, including the selected era, current round, maximum rounds, player name, and scores of all roles.

private void drawLegend(Graphics g) Method: Draws a small legend on the side of the screen that shows which icon belongs to each role and food. It helps the player easily understand what is displayed on the game board.

private void loadEraImages() Method: Loads all images based on the selected era, including animal sprites, food images, and the background. By using the Eraenum, the game automatically switches visuals when a different era is selected.

private Image loadImage(String path) Method: Loads an image from the project resources using the given file path. If the image cannot be found, an error is thrown to prevent the game from running with missing assets.

public void setState (GameState newState) Method: Updates the game panel when a new game state is loaded. It replaces the current state, creates a new controller, refreshes the screen, and requests keyboard focus so the player can continue interacting with the game.

4. Package Game

4.1 Grid Class

Defines the size of the game board and represents the playable area of the game.

public boolean Inside(Position position) Method: Checks whether a given position is inside the grid boundaries. It returns true if the position is valid and false if it lies outside the game area.

4.2 GameState Class

Acts as the central data holder of the game, storing all information needed to control gameplay, evaluate progress, save the game, and display scores.

public String serialize() Method: Converts the current game state into a text format, including era, grid size, round number, animals, and food, so that the game can be saved to a file.

public String roundSummary() Method: Generates a readable summary of the current round, including the era, player name, and the scores of all animals.

4.3 GameController Class

Manages the main game logic, including player actions, AI behavior, collision handling, scoring, round progression, and determining the game winner.

public void movePlayer(Animal player, Position next) Method: Moves the player to the next position if it is inside the grid and checks for possible collisions after the move.

public void playRound() Method: Executes one full game round by updating cooldowns, moving AI animals, checking collisions, increasing the round number, and logging the round summary.

private void moveAIAnimals() Method: Moves all AI-controlled animals based on their roles and behaviours.

private Position fleeFood(Position preyPosition) Method: Calculates a movement direction for the Prey to move away from the nearest food. preyPosition is the current position of the Prey and returns a new position that increases distance from food.

private void logRoundSummary() Method: Logs the current round number and the scores of all animals to the game log file.

private Position chasePlayer(Position from, Position target) Method: Calculates a movement step that moves one position closer to a target and returns the next position toward the target.

private void checkCollisions() Method: Detects interactions between animals and food, updates scores, respawns animals or food when necessary.

private void respawnFood(Food food) Method: Respawns food at a random valid position on the grid.

private boolean samePosition(Position p1, Position p2) Method: Checks whether two positions are the same. And returns true if both positions are equal. It is mainly used during collision detection to determine interactions between animals and food, such as eating food or one animal consuming another.

private boolean isInsideGrid(Position p) Method: Checks whether a position is inside the grid boundaries and returns true if the position is valid.

public boolean isGameOver() Method: Returns true if the maximum number of rounds has been reached.

public String determineWinner() Method: Compares the player score with AI scores and determines the winner and returns a string describing the winner.

private Position preySmartMove(Position preyPosition) Method: Determines the safest and most beneficial move for the Prey by balancing predator distance and food proximity and returns the best calculated next position.

private Position apexSmartChase(Position apexPos) Method: Moves the Apex Predator toward the closest Prey or Predator and returns the next position toward the closest target.
public String determineGameWinner() Method: Determines the final winner of the game based on scores and returns a string describing the final result.

private Position randomPosition() Method: Generates and returns a random valid position inside the grid.
rivate void respawnAnimal(Animal a) Method: Places an animal at a random empty position after it is eaten.

private boolean isEmpty(Position p) Method: Checks whether a given position is free of other animals.

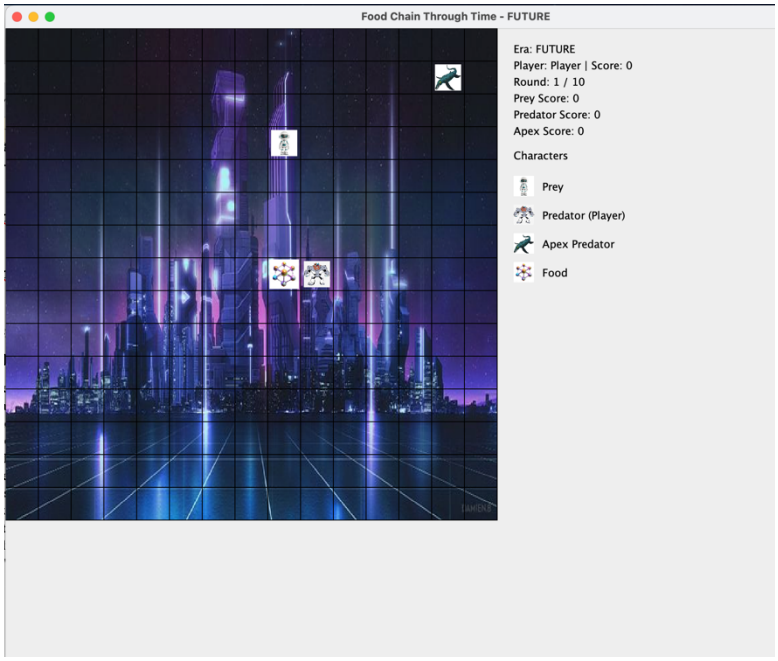
private void resetAfterRound() Method: Resets the game by clearing animals and spawning new ones for a new round.

5 Package App

Main Class

It starts the game, asks the user for game settings. The program first asks the user to enter a username using a dialog box and stores the input in the variable username. If the user presses cancel or leaves the input empty, the program assigns the default name "Player" to avoid having an empty username. Next, the user is asked to select an era (Past, Present, or Future) using another dialog window, and the selected value is stored in the variable era. If the user does not select, the program automatically sets the era to Present. After that, the program defines possible grid size options (10x10, 15x15, 20x20) and shows them to the user in a dialog box. The selected option is stored as a string in gridChoice. Since the grid size must be a number to create the game grid, a switch statement converts the selected string into an integer value and stores it in gridSize.

Figure 9.



6. Package File

6.1 FoodChainLoader Class

Loads the initial game setup for a selected era from a text file. It creates animals and food dynamically based on external configuration data.

public static void loadEra(Era era, GameState state, String username) Method: This method reads an era-specific text file and initializes the game by spawning animals and food at random positions. It allows the game configuration to change without modifying the code.

private static Position randomPos(GameState state) Method: Generates a random position inside the game grid and returns a position inside the grid boundries. It is mainly used when spawning animals and food so that their starting locations are different each time the game begins.

6.2 GameLogger Class

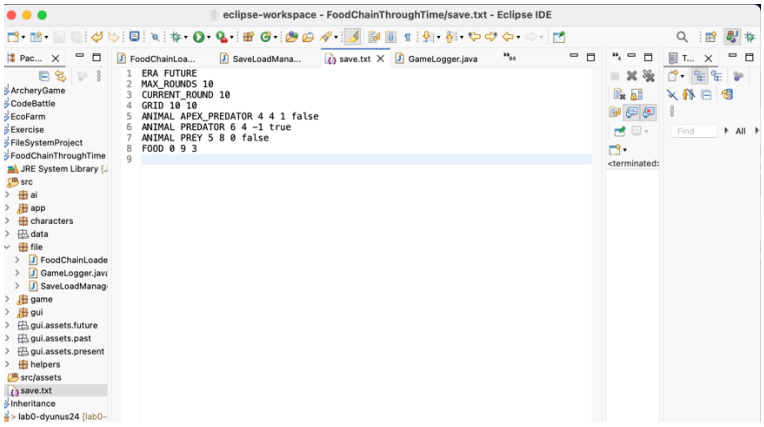
Records important game events into a text file named gameLog.txt.

6.3 SaveLoadManager Class

It is responsible for saving the current game state to a file and loading it back later. This allows the player to continue the game from where it was left.

public static void save(GameState state) Method: This method writes all important game information into a text file called save.txt. It saves the selected era, grid size, current round, animals with their positions and scores, and food positions. This ensures that the entire game state is stored safely.

Figure 10.



public static GameState load(String filename) Method: Reads the saved game file and rebuilds the game exactly as it was. It restores the grid, era, round number, animals, food, and player information, allowing the game to continue without losing progress.

7. Package AI

The ai package contains the artificial intelligence logic responsible for autonomous decision-making of non-player-controlled roles. This package encapsulates movement strategies and evaluation heuristics, allowing AI behaviour to remain independent from both the model and GUI layers.

7.1 PreyAI Class

The PreyAI class controls the movement behaviour of Prey entities. Its primary goal is to maximize survival while opportunistically moving toward food. Prey movement is determined by evaluating all possible neighbouring positions and assigning a score to each position based on environmental risk and reward.

decideMove(Prey prey, GameState state) Method:

The decideMove method determines the most suitable next position for a Prey based on the current game state. It starts by obtaining the Prey's current position (current), which serves as the reference point for evaluating movement options. All valid neighbouring positions are generated and stored in possibleMoves by iterating over small offsets in the x and y directions, allowing movement in eight directions as well as remaining stationary. Each

candidate position is checked to ensure it lies within the grid boundaries. The variable minPredatorDist stores the minimum Manhattan distance to any Predator or Apex Predator, while minFoodDist stores the minimum distance to the nearest food item. Both values are initialized using Integer.MAX_VALUE to allow proper comparison during evaluation. Based on these distances, a score is calculated that heavily penalizes positions close to predators and rewards safer positions, with proximity to food becoming more important when predator risk is low. The variables bestScore and bestMove track the highest score encountered and the corresponding position. After all candidate positions are evaluated, the method returns bestMove as the optimal move for the Prey. If no optimal move is found, a random valid position is selected as a fallback. This approach enables adaptive and efficient AI behaviour without relying on complex pathfinding algorithms.

7.2 ApexPredatorAI Class

The ApexPredatorAI class defines the movement strategy of the Apex Predator, whose primary objective is to pursue and eliminate Prey. Unlike the Prey, which evaluates multiple environmental factors, the Apex Predator follows a direct chase-based strategy that prioritizes the closest target**.**

decideMove(ApexPredator apex, GameState state) Method: Controls how the Apex Predator moves in the game. It first gets the current position of the Apex Predator and then searches for the closest Prey by checking all animals in the game state. The distance to each Prey is calculated using Manhattan distance, and the nearest one is selected as the target. If no Prey exists, the Apex Predator stays in its current position. After selecting a target, the method calculates the direction of movement using stepX and stepY, which determine how the Apex Predator should move one grid cell closer to the target. A new position is created and checked to ensure it is inside the grid. If the position is valid, it is returned as the next move; otherwise, the Apex Predator does not move. This simple chase-based logic makes the Apex Predator a constant threat while keeping the AI efficient and easy to understand.

References

- **Vecteezy.** ***Wolf PNG Transparent Background.*** **https://www.vecteezy.com/png/12893848-wolf-transparent-background**

- **DeviantArt.** ***Leviathan PNG.*** **https://www.deviantart.com/gothsavagebowserx/art/Leviathan-PNG-880329137**
- **TopPNG.** ***Teen Titans Go Cyborg PNG.*** **https://toppng.com/free-image/teen-titans-go-cyborg-PNG-free-PNG-Images_46739**
- **PNGTree.** ***Robot Images.*** **https://pngtree.com/so/robot**
- **Vecteezy.** ***Abstract Network Illustration.*** **https://www.vecteezy.com/png/58630377-wonderful-minimalist-a-floating-network-of-interconnected-nodes-pulsating-with-energy-each-node-emitting-a-different-colored-light-lit-by-an-internal-glow-abstract-style-4k**
- **PNGIMG.** ***Dinosaur PNG.*** **https://pngimg.com/image/4933**
- **Freepik.** ***T-Rex Dinosaur PNG.*** **https://www.freepik.com/free-photos-vectors/t-rex-dinosaur-png**
- Çevik, M. T. (n.d.). Java Swing ve GUI Programlama: Kullanıcı Arayüzleri Oluşturmanın Temel Rehberi. Retrieved from https://muhammedtalhacevik.medium.com/java-swing-ve-gui-programlama-kullanıcı-arayüzleri-oluşturmanın-temel-rehberi-by-m