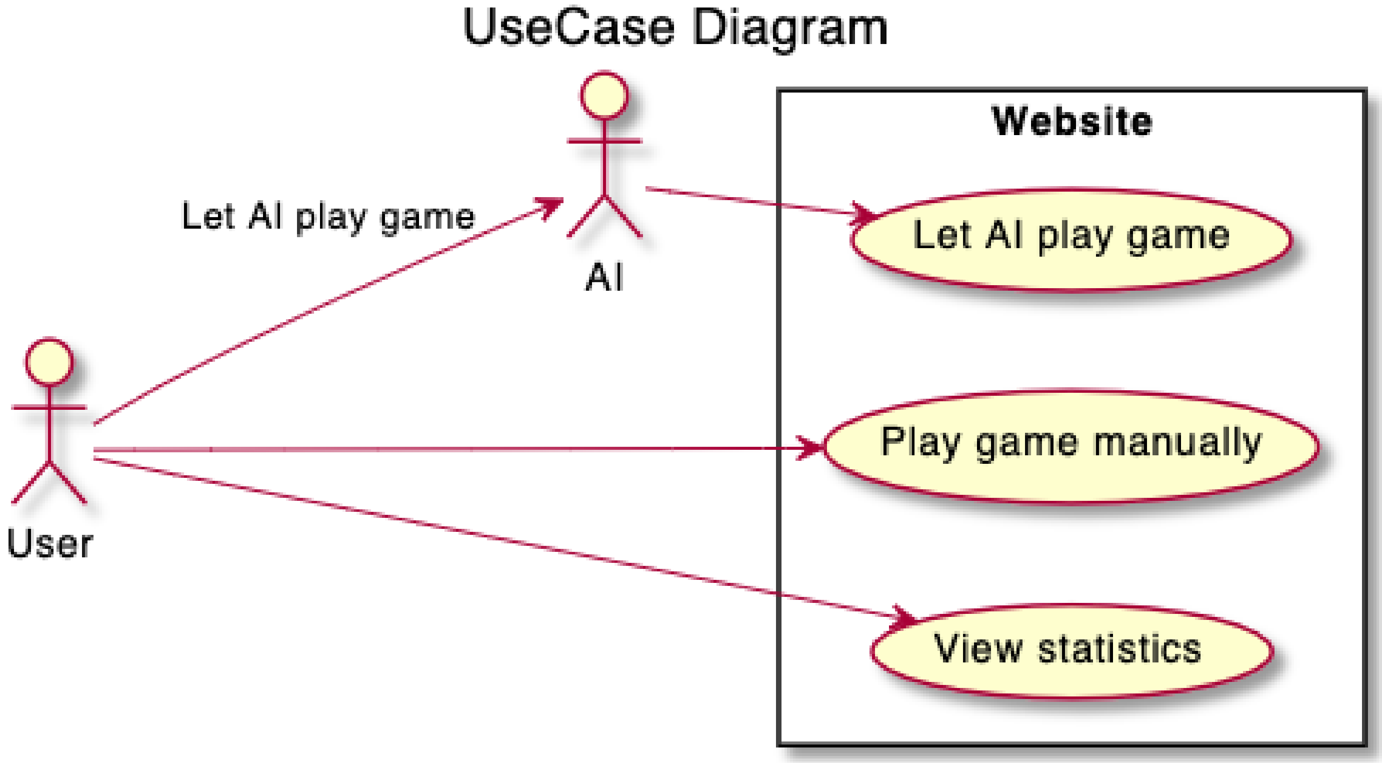
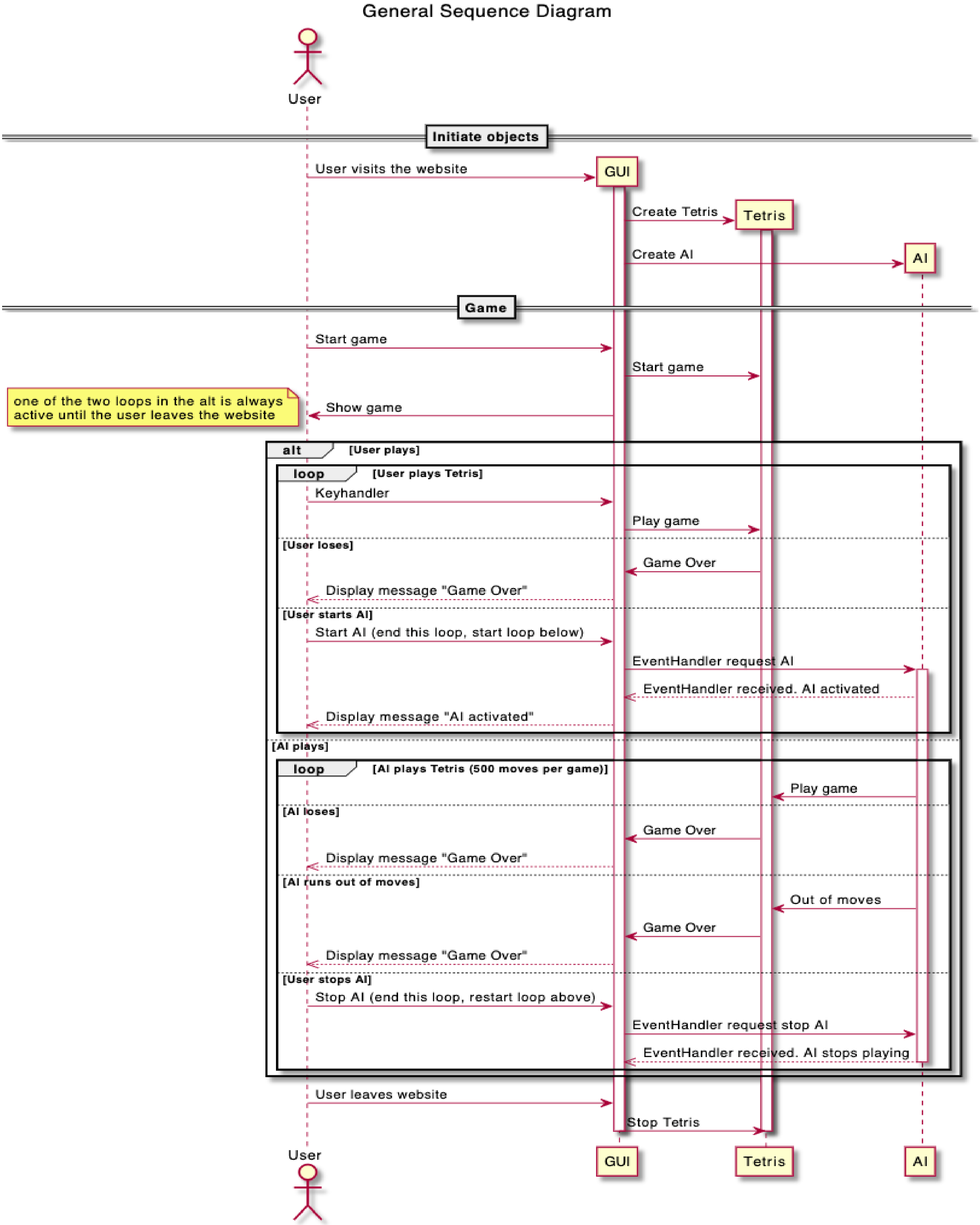
# Tetris bot

AI is being used more and more frequently in various sectors. Working with it can be challenging and difficult, so the team decided to create a project which will demonstrate the use of AI in a game of Tetris. The idea behind it is to make sure that the AI grows and evolves the more games it plays. Keeping this goal in mind, the team decided to use a genetic algorithm as the base for the AI.

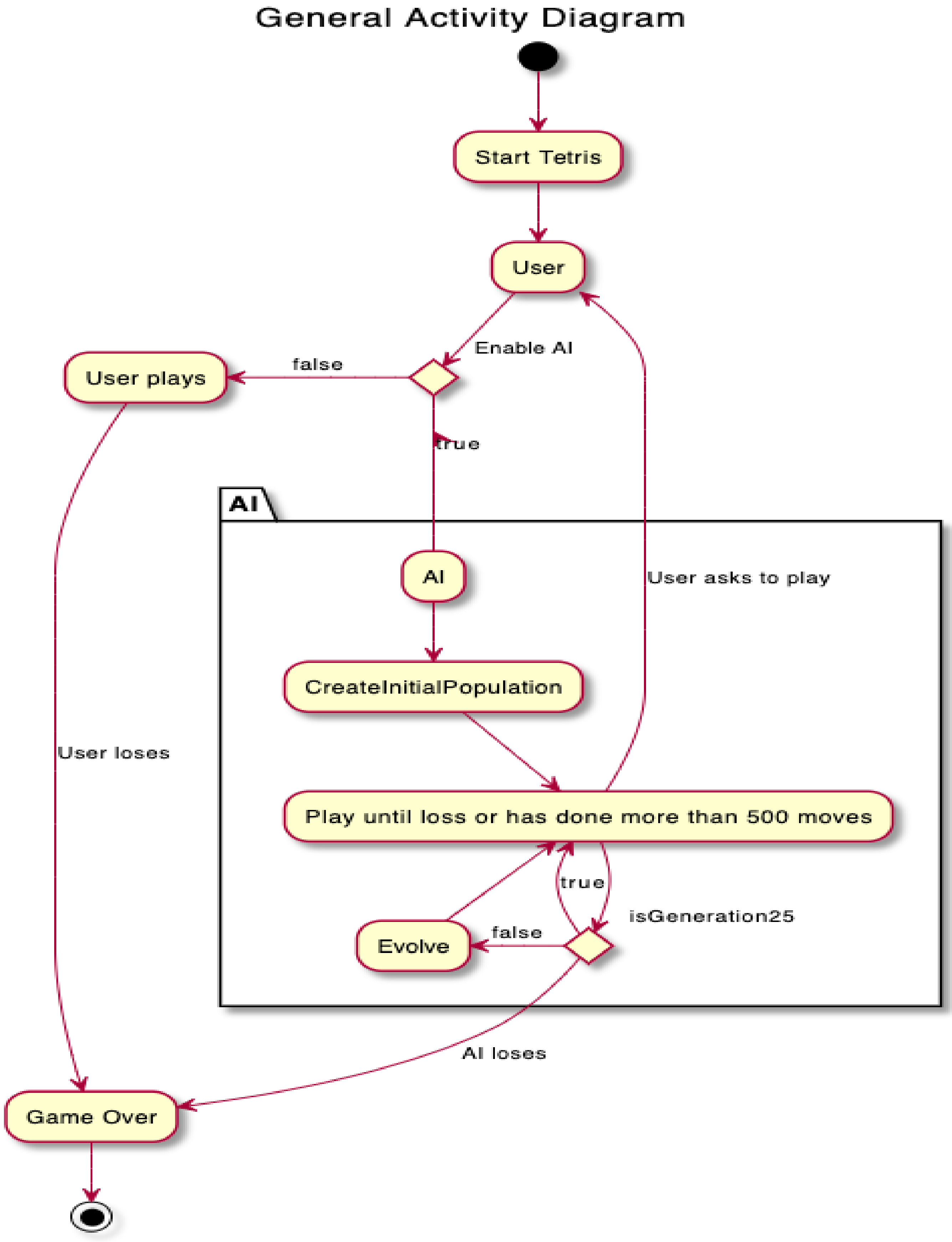
To begin with the analysis, the creation of a Use Case Diagram seemed to be a good starting point. Here, the general demonstration of the program is shown without diving deep in the backend.



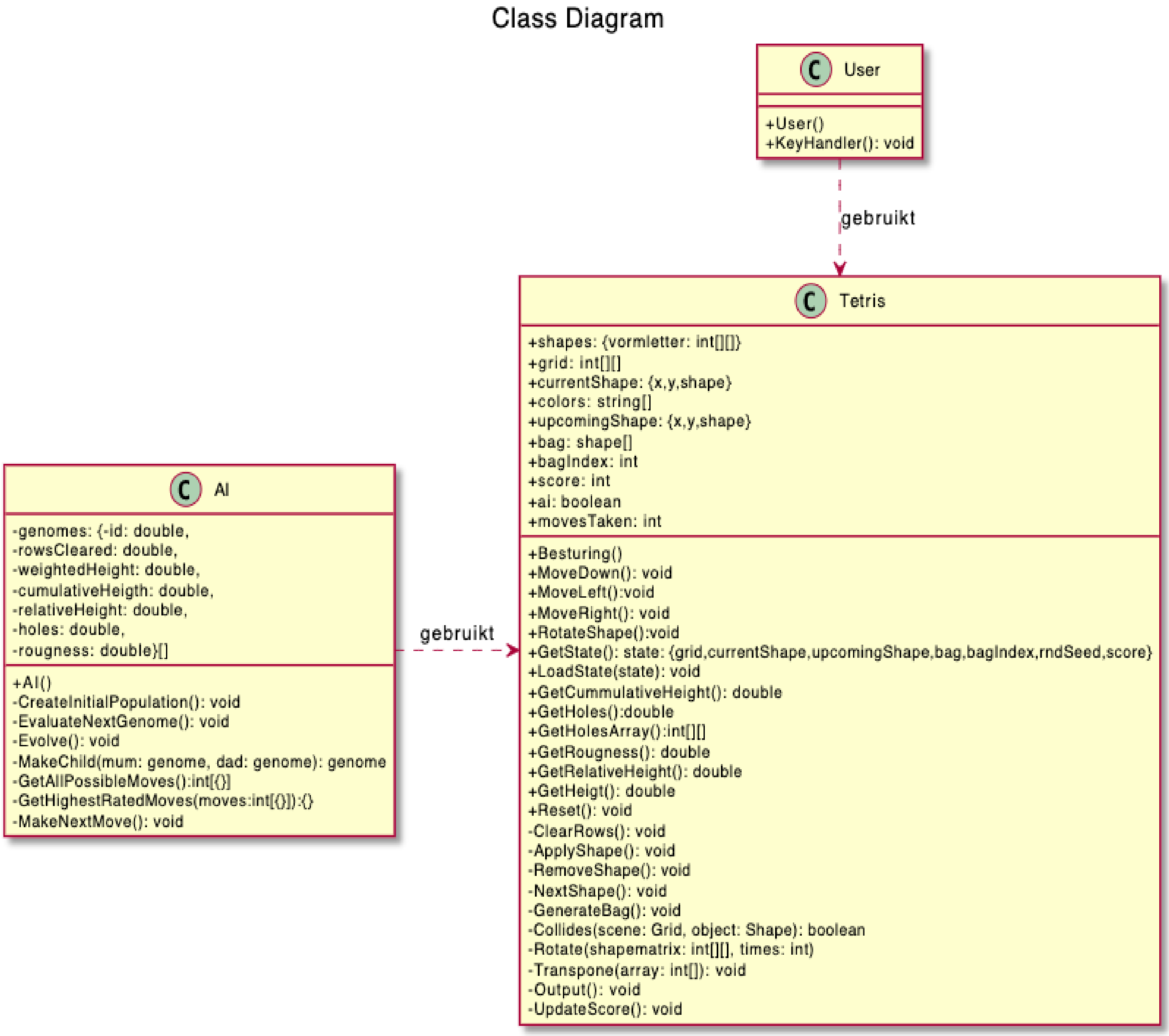
The following diagram demonstrates the General Sequence of the game. This diagram shows when various components of the game are being activated and under which circumstances the game can be terminated.


General Sequence Diagram
User
Initiate objects
User visits the website
GUI
Create Tetris
Tetris
Create AI
AI
Game
Start game
Start game
one of the two loops in the alt is always active until the user leaves the website
Show game
alt
[User plays]
loop
[User plays Tetris]
Keyhandler
Play game
[User loses]
Game Over
Display message "Game Over"
[User starts AI]
Start AI (end this loop, start loop below)
EventHandler request AI
EventHandler received. AI activated
Display message "AI activated"
[AI plays]
loop
[AI plays Tetris (500 moves per game)]
Play game
[AI loses]
Game Over
Display message "Game Over"
[AI runs out of moves]
Out of moves
Game Over
Display message "Game Over"
[User stops AI]
Stop AI (end this loop, restart loop above)
EventHandler request stop AI
EventHandler received. AI stops playing
User leaves website
Stop Tetris
User
GUI
Tetris
AI

In the General Activity Diagram, the general working pattern of the game is shown in a comprehensive way. In the AI column, the condition under which the AI evolves and grows are shown. The game is played until the AI loses or until a gene of the AI has finished its 500 moves. This results in the next gene starting to play the game. After all 50 genes in a generation have played, the AI evolves and moves on to the next, better generation.

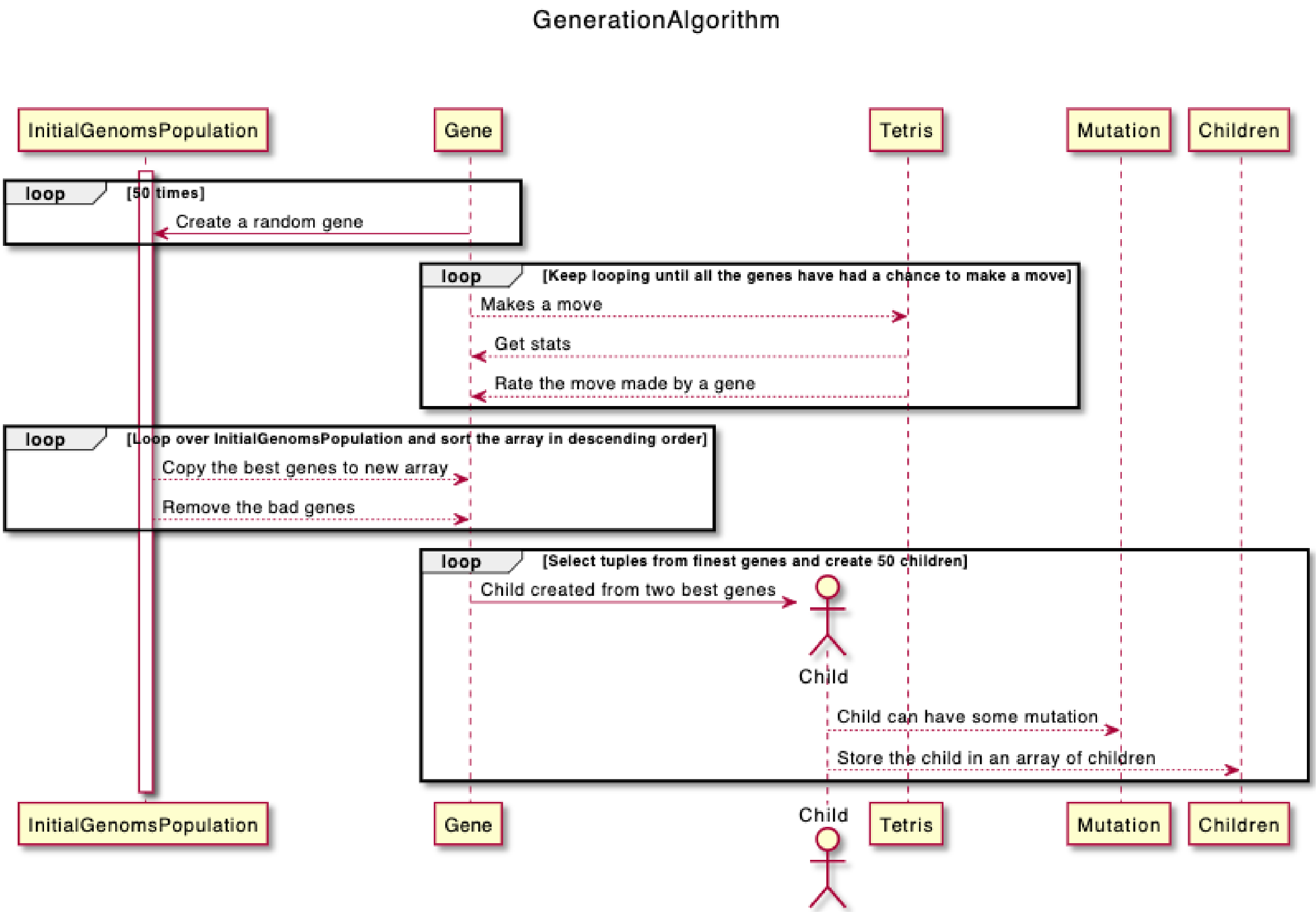


The Class Diagram shows the various classes and methods that might be needed to code the game. This doesn't include the UI (User Interface) or UX (User Experience). The team decided to create three major classes (AI, User and Tetris). AI class contains the methods and parameters needed for the AI. Tetris contains the general methods and parameters for the game including the functions of AI. User Class contains the methods and parameters so that the user can play the game (instead of the AI).



A more in-depth view of the AI’s Genetic Algorithm:

50 genes are generated per generation. Each gene gets the chance to play the game. The moves made by each gene get rated. Once the genes are done with their moves, they get sorted in ascending order based on their respective rating. The ones with the best ratings are selected to create children for the next generation. There is also the possibility of a child mutating their genes. At the end of the cycle a new generation of 50 genes is created out of the best scoring ones from the previous generation, at which point the cycle begins again. This way the AI learns and evolves.


GenerationAlgorithm
InitialGenomsPopulation
Gene
Tetris
Mutation
Children
loop
[50 times]
Create a random gene
loop
[Keep looping until all the genes have had a chance to make a move]
Makes a move
Get stats
Rate the move made by a gene
loop
[Loop over InitialGenomsPopulation and sort the array in descending order]
Copy the best genes to new array
Remove the bad genes
loop
[Select tuples from finest genes and create 50 children]
Child created from two best genes
Child
Child can have some mutation
Store the child in an array of children
InitialGenomsPopulation
Gene
Child
Tetris
Mutation
Children