Санкт-Петербургский политехнический университет Петра Великого

Кафедра компьютерных систем и программных технологий

**Отчёт по проекту**

**Дисциплина**: Алгоритмы и структуры данных

**Тема:** Решатель тетриса

Выполнил студент гр. 3530901/90003 Т. Р. Иванов

(подпись)

Преподаватель М.Х. Ахин

(подпись)

“ ” 2020 г.

Санкт-Петербург

2020

**Оглавление**

[**1. Техническое задание** 3](#_Toc57231376)

[**2. Метод решения** 10](#_Toc57231377)

[**3. Описание классов и методов** 12](#_Toc57231378)

[**4. Работа программы** 20](#_Toc57231379)

**1. Техническое задание**

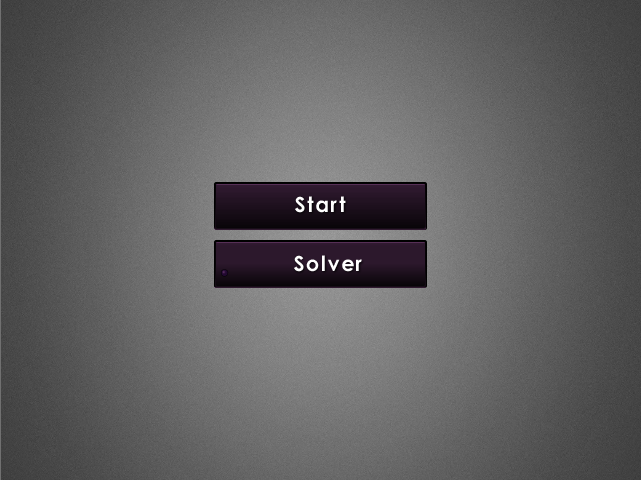
**Общие сведения**

* Приложение реализовано на LibGDX
* На первом экране приложения присутствуют следующие элементы:
  + Изображение на фоне
  + Кнопка «Start», нажатие которой переводит приложение на экран игры
  + Кнопка «Solver». Нажатие на нее меняет текущее состояние решателя. Если индикатор на кнопке горит фиолетовым светом – решатель работает, если индикатор не горит – то решатель выключен. Если решатель включен, то нажатие на кнопку «Start» приводит не к игровому экрану, а к экрану настроек решателя.
* На экране настроек решателя есть:
  + Изображение на фоне
  + Поле ввода, на котором написана подсказка “Set Number of Lines”. При нажатии на поле подсказка пропадает и от пользователя требуется ввести число – количество линий, которое решатель должен «очистить» для победы. В поле можно ввести любое число от 1 до 99999
  + Кнопка «Start». При нажатии выводит на экран игры. Кнопка недоступна, пока пользователь не введет в поле ввода какое-либо число.
* Игровой экран немного изменяется, в зависимости от того, включен ли решатель или нет
  + Общие элементы:
    - Изображение на фоне
    - Игровое поле, представляющее собой массив клеток 20x30
    - Падающие фигурки, также представляющие собой массив клеток
    - Небольшое поле справа от игрового, показывающее какая фигурка должна упасть после текущей
    - Поле «Lines» отображающие текущий счет
    - Кнопка «Pause». Нажатие на нее останавливает игру. Повторное нажатие возобновляет игру.
    - Кнопка «Reset». Возвращает игровой экран в стартовое состояние.
    - Кнопка «Exit». Переводит игру на стартовый экран
  + Отличающиеся элементы:
    - Цвет фигурок. Если решатель выключен, то фигурки будут синего цвета, а если включен – красного.
    - Поле «Lines». Когда решатель выключен, то при старте поле отображает 0. При сборе полной линии счет увеличивается на 1 за каждую очищенную линию. Если же решатель включен, то поле «Lines» будет отображать то число, чтобы было набрано в поле на экране настроек решателя. В этом случае, при сборе линии счет будет уменьшаться на 1 за каждую очищенную линию.
    - Если решатель выключен, то фигурками можно управлять с помощью кнопок «WASD» на клавиатуре. Кнопки «A» и «D» передвигают фигурку на одну клетку влево или вправо соответственно. Кнопка «W» поворачивает фигурку против часовой стрелки. Если фигурку находится у границы поля и ей не хватает места чтобы повернуться против часов стрелки, фигурка поворачивается по часовой стрелке. Кнопка «S» увеличивает скорость падения фигурки. Если решатель включен, то управление фигурками пользователю недоступно.
    - Скорость падения фигурок: фигурки падают быстрее, если решатель включен.
* Если фигурка выходит за верхнюю границу игрового поля, или же решатель набирает нужное ему количество линий для победы, то игра переходит на экран «конца игры». На экране присутствуют следующие элементы:
  + Изображение на фоне
  + Текст «Your Score:», отображающий счет, на котором закончилась игра
  + Кнопка «Restart», которая в случае выключенного решателя сразу же переводит приложение на игровой экран. Если же решатель включен, то нажатие на кнопку переведет игру на экран настроек решателя.
  + Кнопка «Menu». Переводит на стартовый экран.

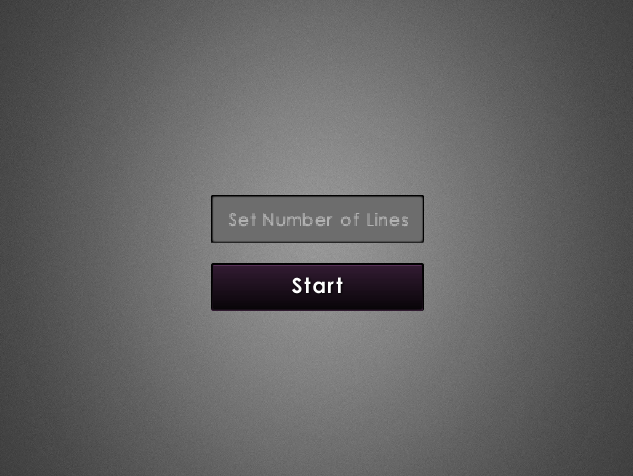
**GitHub**

<https://github.com/iTteruya/tetris_solver>

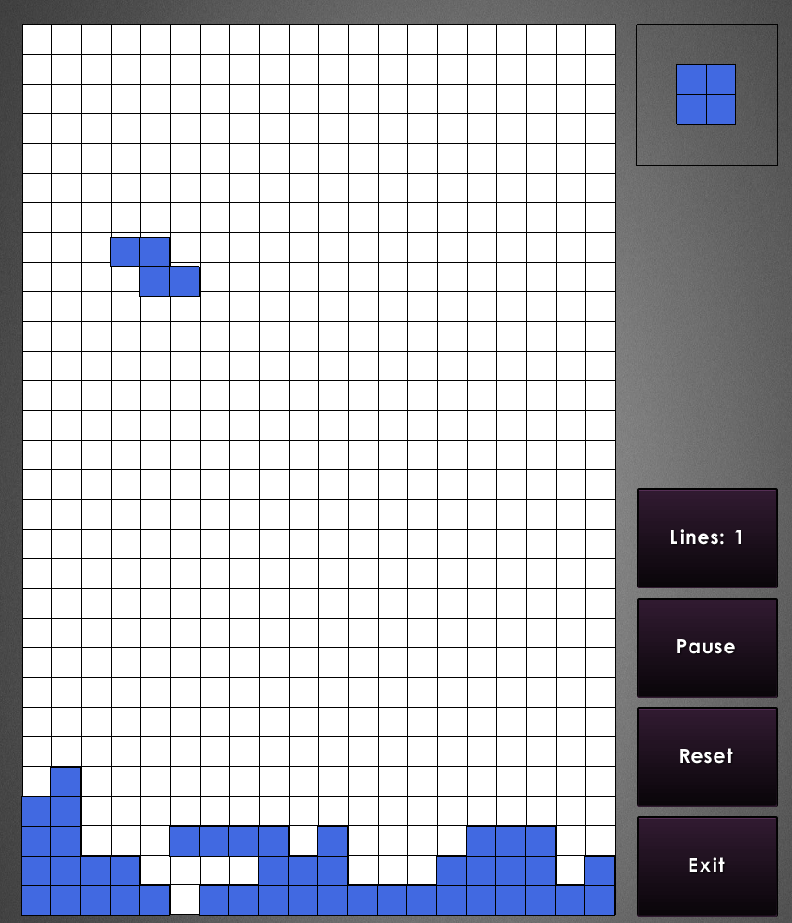
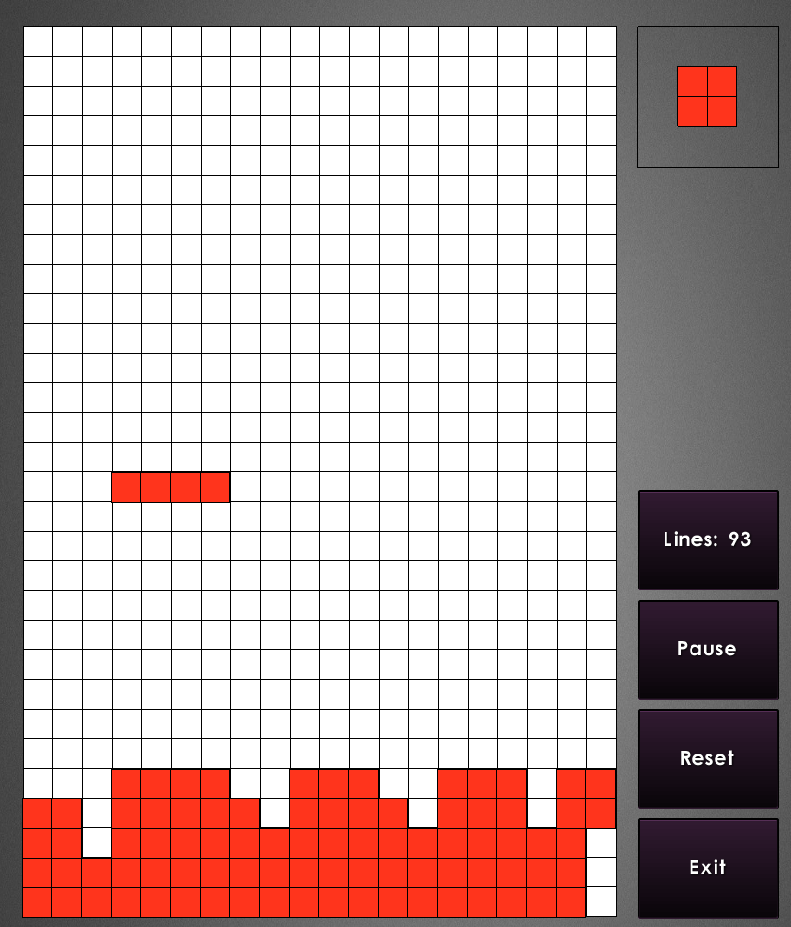
**Скриншоты приложения**



Стартовый экран



Экран настроек решателя

Игровой экран, решатель выключен 

Игровой экран, решатель включен



Экран «конца игры»

**2. Метод решения**

В основе работы ИИ лежит идея генетического алгоритм, схожего с A\*.

Суть генетический алгоритма заключается в том, что путем случайного подбора, комбинирования и вариации искомых параметров находится наилучшее решение, данным образом симулируя «естественный отбор».

Некоторым, обычно случайным, образом создаётся множество генотипов начальной популяции. Они оцениваются с использованием «функции приспособленности», в результате чего с каждым генотипом ассоциируется определённое значение («приспособленность»), которое определяет насколько хорошо фенотип, им описываемый, решает поставленную задачу.

Из полученного множества решений («поколения») с учётом значения «приспособленности» выбираются решения, к которым применяются «генетические операторы» (в большинстве случаев «скрещивание» — crossover и «мутация» — mutation), результатом чего является получение новых решений. Для них также вычисляется значение приспособленности, и затем производится отбор («селекция») лучших решений в следующее поколение.

Этот набор действий повторяется итеративно, так моделируется «эволюционный процесс», продолжающийся несколько жизненных циклов (поколений), пока не будет выполнен критерий остановки алгоритма.

**Алгоритм работает следующим образом:**

1. Смотрим на текущую фигурку и симулируем все возможные комбинации (позиции и повороты) для нее.
2. Даем оценку каждой позиции
3. Передвигаем фигурку на позицию с наибольшей оценкой и повторяем.

Для оценивания позиции используются следующие критерии:

1. Количество «дыр» - мест, куда можно поставить фигурки
2. Количество заполненных линий
3. Высота – индекс клетки фигурки, которая находится выше всех остальных на поле.

Выбор лучшего хода и сравнивание оценок происходит следующим образом:

1. Попарно сравниваются между собой оценки всех возможных ходов.
2. Из первого набора оценок (кол-во «дыр», заполненные линии, высота) вычитается второй набор.
3. Если количество «дыр» в первом наборе меньше, чем во втором, то данный считается, что этот ход лучше второго
4. Если количество «дыр» равное, то сравнивается количество заполненных линий. Если количество заполненных линий в первом наборе больше, чем во втором, то данный ход считается лучше.
5. Если количество линий так же равно, то сравнивается высота. Если, максимальная высота первого набора меньше, чем второго, то данный ход считается лучше.
6. Во всех остальных случаях второй ход считается лучше первого.

**3. Описание классов и методов**

**Класс Cell:**

Отражает характеристики клетки (позиция и размеры)

**Методы класса Cell:**

setPosition(int x, int y):

Задет позицию клетки

getX():

Возвращает x-координату клетки

getY():

Возвращает y-координату клетки

getWidth():

Возвращает ширину клетки

getHeight():

Возвращает высоту клетки

**Класс GameField:**

Отвечает за логику и прорисовку игрового поля. Поле представляется как массив единиц и нулей. Если клетка должна быть закрашена, то в массиве на ее месте находится единица, если нет – то нуль.

**Методы класса GameField:**

reset():

Обнуляет игровое поле

addFigure(Piece piece):

Добавляет фигурку на поле

update(Piece piece):

Обновляет состояние поля. Выдает true пока фигурка не окажется на «полу»

clearLine(boolean solverON):

Определяет есть ли на поле заполненные линии, и в случае их существования «очищает» их. Также за каждую очищенную линию изменяет счет поля, увеличивая на один, а если решатель выключен – уменьшая на один.

gameIsNotOver():

Определяет состояние игры – продолжается ли еще игра, или какая-то из фигурок вышла за пределы поля и игрок проиграл

getScore():

Возвращает текущий счет поля

setScore(int score):

Позволяет установить счет поля

getMatrix():

Возвращает матрицу поля

getRows():

Возвращает количество строк игрового поля

getCols():

Возвращает количество столбцов игрового поля

draw(Batch, float parentAlpha):

Отвечает за отрисовку игрового поля

**Класс Piece:**

Отвечает за логику и отрисовку игровых фигурок

Класс Piece содержит подкласс-перечисление Type

Класс **enum Type**:

Перечисляет все возможные типы фигурок

**Методы класса Piece:**

init(int i, int j):

Выполняется при инициализации. Задает исходную позицию фигурки и случайным образом присваивает ей один из типов enum-класса Type и соответствующую этому типу матрицу фигурки. Помимо матрицы фигурке также присваиваются координаты deltaX и deltaY. Эти координаты нужны, чтобы все фигурки отражались одинаково внутри поля следующих фигур (чтобы фигурка находилась ровно в центре этого поля). Также случайным образом фигурка может быть «отзеркалена».

shift(int direction, int[][] matrix):

Перемещает элементы матрицы фигурки в заданном направлении (direction): -1 – влево, 1 – вправо, -2 – вниз, 2 – вверх.

moveToCorner(int[][] matrix):

Сдвигает матрицу фигурки в левый нижний угол

mirror():

«Отзеркаливает» фигурку

rotate():

Отвечает за поворот фигурки

canPlace(int col, int[][] matrix):

Определяет можно ли поместить фигурку в данный столбец (col) игрового поля.

getX():

Возвращает x-координату клетки

getY():

Возвращает y-координату клетки

changeX(int inc):

Меняет координату x на заданную величину (inc)

changeY(int inc):

Меняет координату y на заданную величину (inc)

moveRight():

Перемещает фигурку на клетку вправо на игровом поле

fall():

Перемещает фигурку на одну клетку вниз на игровом поле

getCurrentRow():

Возвращает текущую строку, на которой находится фигурка

getCurrentColumn():

Возвращает текущий столбец, на котором находится фигурка

getMatrix():

Возвращает матрицу фигурки

set(Piece piece):

Присваивает фигурке характеристики другой фигурки (тип, позиция, матрица)

getCells():

Создает список из 4 клеток, которые потом будут использованы для отрисовки фигурки

renderCell (Cell cell, ShapeRenderer renderer, Color color, boolean nextPiece):

Отвечает за рендер (отрисовку) клетки

renderPiece(Piece piece, ShapeRenderer renderer, Color color, boolean nextPiece):

Отвечает за отрисовку фигурки. Разбивает фигурку на 4 клетки, задает им позиции и по отдельности «рендерит» каждую из этих 4-х клеток

fieldStateRender(GameField gameField, ShapeRenderer renderer, Color color):

Заново отрисовывает поле, после того как фигурка полностью упадет

**Класс Strategist:**

Класс нужный для реализации работы алгоритмы. Дает оценку состояния поля и позволяет сравнивать с другими состояниями игрового поля.

**Методы класса Strategist:**

getPiece():

Возвращает фигурку, симуляция хода которой создало текущее состояние поле

getScoreList():

Возвращает список оценок текущего состояния поля

getBetter(Strategist newScores):

Сравнивает две оценки и выдают лучшую

setFullLineScore():

Дает оценку заполненных линий поля. Из количества заполненных линий текущего состояния поле вычитается количество заполненных линий прошлого состояния поля

countLines(int[][] matrix):

Находит количество заполненных линий на поле

setHeightScore():

Дает оценку высоты поля. Из высоты прошлого состояния поля вычитается высота текущего состояния поля

getHeight(int[][] matrix):

Находит высоту поля

setHolesScore():

Дает оценку «дыр» поля. Из количества «дыр» прошлого состояния поля вычитается количество «дыр» текущего состояния поля

countHoles(int[][] matrix):

Находит количество «дыр» поля

**Класс AbstractScreen:**

Абстрактный класс, используется для реализации всех остальных экранов приложения

**Класс GameScreen:**

Класс, отвечающий за работу игрового экрана

**Класс** **MenuScreen**:

Класс, отвечающий за работу стартового экрана

**Класс** **SolverSettingsScreen:**

Класс, отвечающий за работу экрана настроек решателя

**Класс** **EndScreen:**

Класс, отвечающий за работу экрана «конца игры»

**У классов GameScreen, MenuScreen, EndScreen, SolverSettingsScreen есть много общих методов:**

show():

Отвечает за то, какие элементы должны быть отображены на экране

render(float delta):

Отвечает за отрисовку элементов поля

resize(int width, int height):

Отвечает за перерисовку поля при изменении разрешения (растягивание экрана)

dispose():

Избавляется от элементов экрана при переходе с него

**Так же, у класса GameScreen есть один уникальный метод и один подкласс:**

**Подкласс AI:**

Отвечает за работу ИИ

**Методы класса AI:**

choose(GameField gameField, Piece piece):

Выбирает лучший ход (на какое место должна упасть фигурка и как она должна быть повернута)

play():

Метод реализует ход ИИ

setSpeed(int speed):

Позволяет задать скорость падения фигурок

**Уникальный метод класса GameScreen:**

renderStage():

Отвечает за отрисовку игрового поля и фигурок на игровом экране

**Класс TetrisSolver:**

Главный класс приложения

**Методы класс TetrisSolver:**

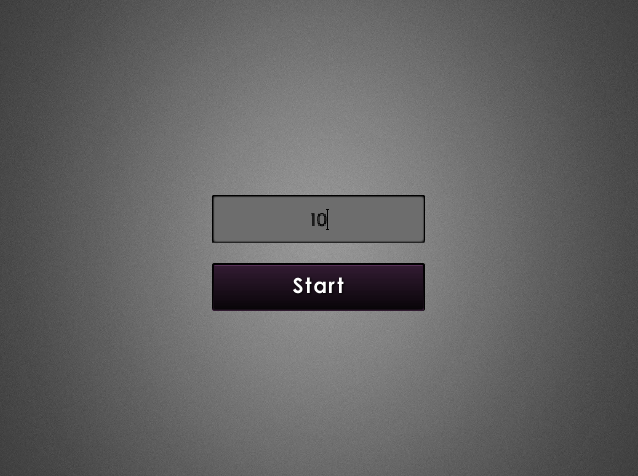
create():

«Создает» стартовый экран приложения

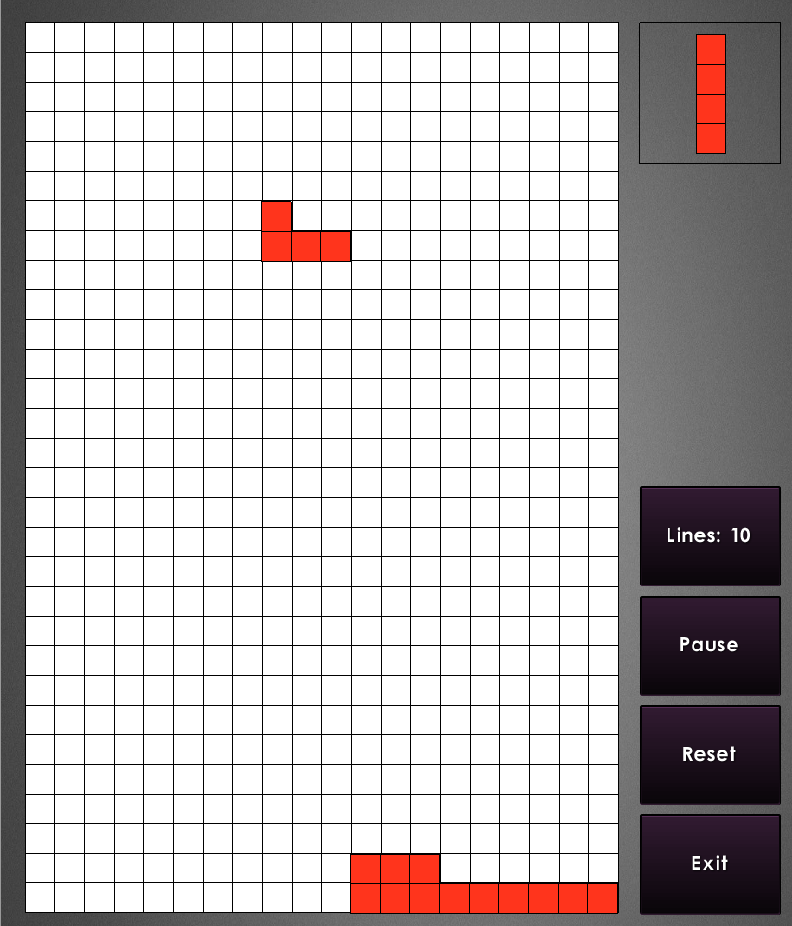
dispose():

Избавляется от всех ресурсов приложения при выходе из него

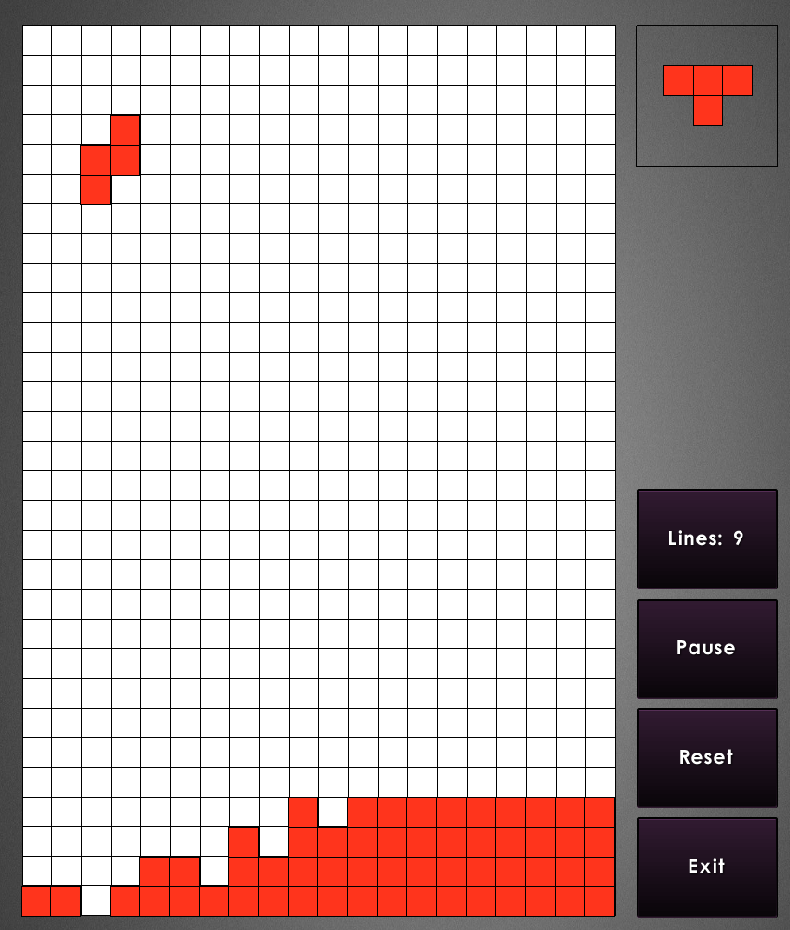
**4. Работа программы**

****

Задаем количество линий, которое решатель должен набрать для победы

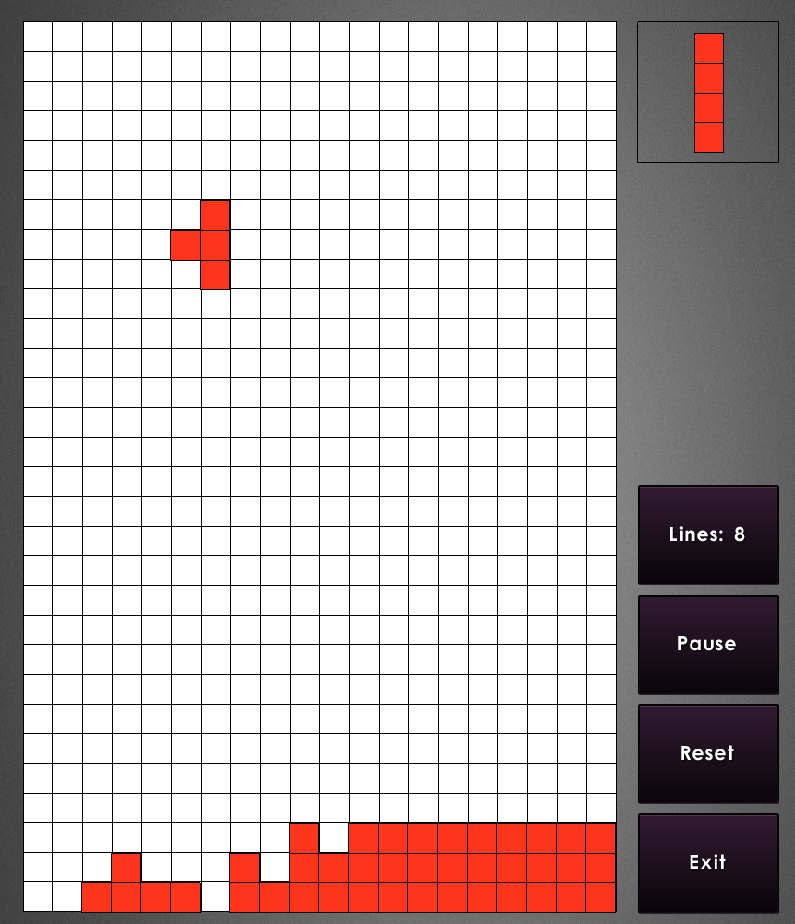


ИИ начинает игру

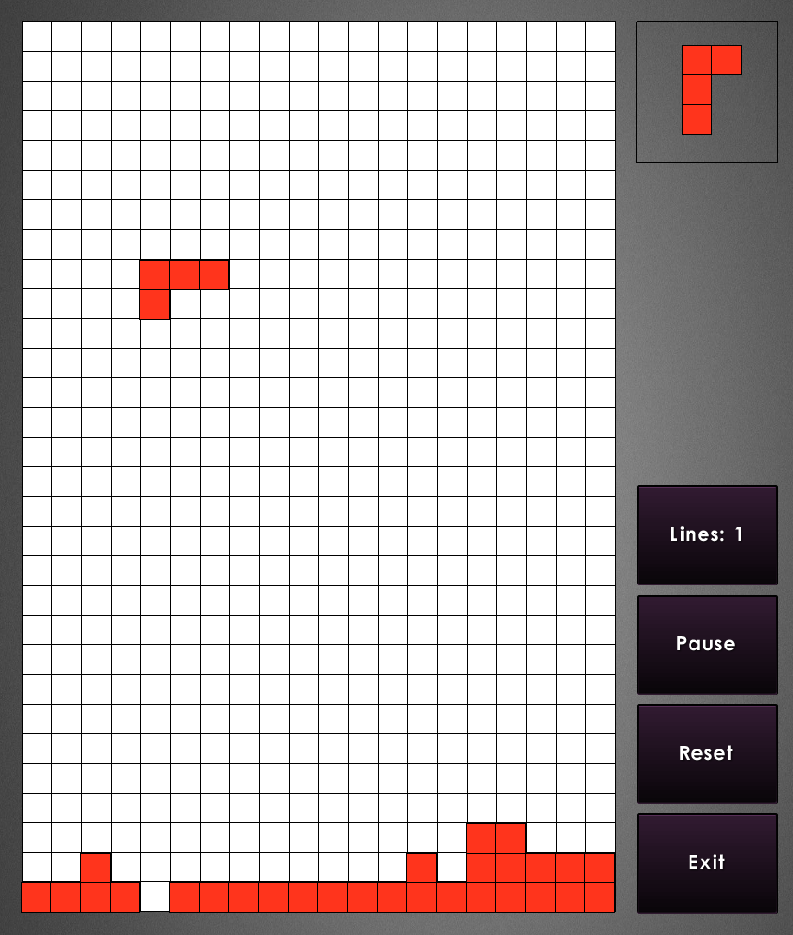


ИИ «очистил» одну линию, счет уменьшился

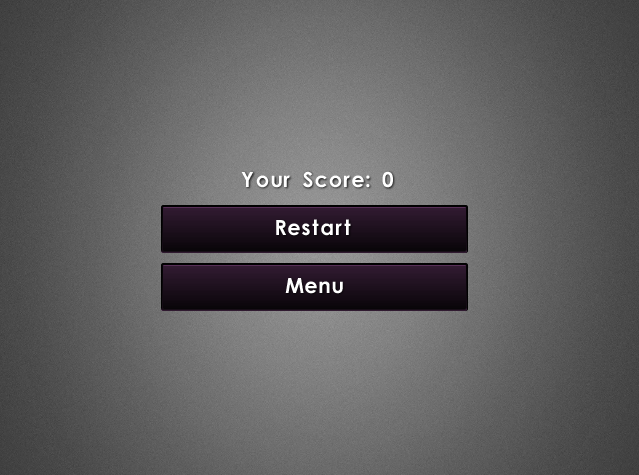
Падает следующая фигурка, которая должна заполнить нижнюю линию



Так и произошло. Фигурка заполнила линию, и эта линия была «очищена». Счет уменьшился на единицу



ИИ продолжает успешно играть, ему остается заполнить еще одну линию



Линия была успешно заполнена, и игра закончилась.

Решатель справился со своей задачей