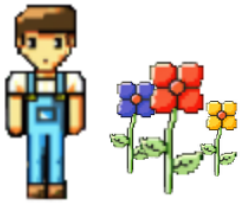


Garden Simulation

Katarzyna Idasz, Bartosz Kawiak

Spis treści

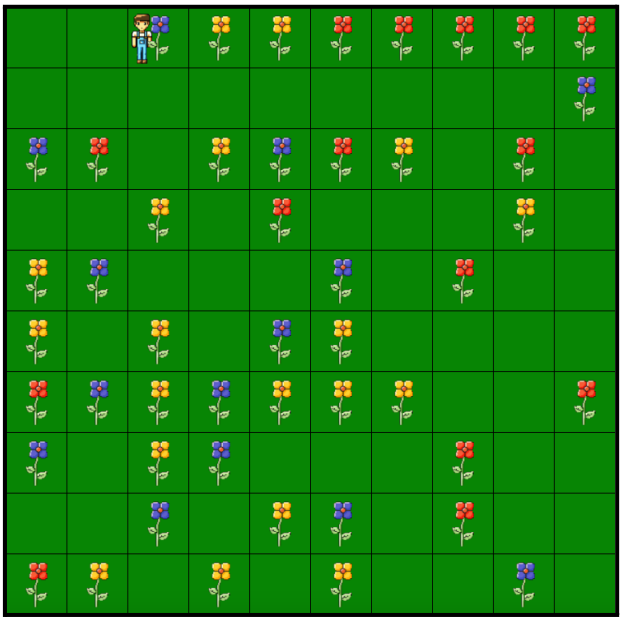
Skład grupy

Katarzyna Idasz - *lider*

Bartosz Kawiak

Opis symulacji

Symulacja przedstawia ogrodnika, który zajmuje się ogrodem. W ogrodzie są trzy rodzaje kwiatków - czerwony, niebieski i żółty. Na zdrowie kwiatków mają wpływ nawodnienie, insekty oraz chwasty.



Szkodniki

Symulacja posiada dwa rodzaje szkodników: insekty oraz chwasty. Zabierają one zdrowie kwiatkom, a ilość obrażeń zależy od tolerancji kwiatka na nie.

Kwiaty ❀

Każdy kwiat ma identyczną maksymalną ilość Hp oraz nawodnienia. Różni je tolerancja na insekty, chwasty i brak wody.

Czerwony kwiat otrzymuje:

- brak wody - 3 obrażenia
- insekty - 2 obrażenia
- chwasty - 2 obrażenia

Żółty kwiat otrzymuje:

- brak wody - 5 obrażenia
- insekty - 2 obrażenia
- chwasty - 1 obrażenia

Niebieski kwiat otrzymuje:

- brak wody - 7 obrażenia
- insekty - 1 obrażenia
- chwasty - 2 obrażenia

Ogrodnik 

Ogrodnik jest odpowiedzialny za nawadnianie kwiatów, odbudowywanie ich punktów zdrowia oraz usuwanie insektów oraz chwastów.

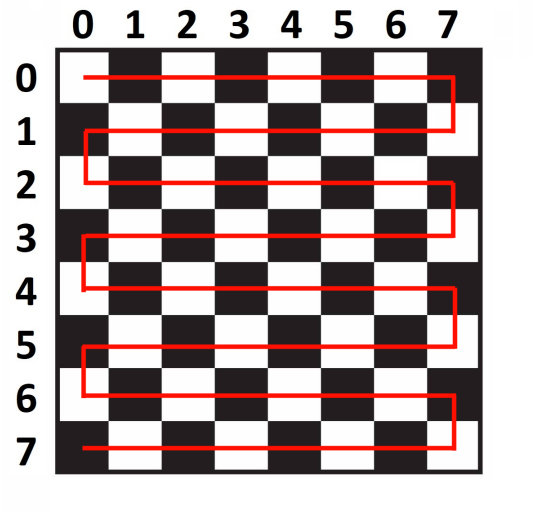
Mamy dwa typy ogrodników:

- **PathFollowingGardener** zawiera w sobie metody poruszania się, które są obojętne na stan ogrodu - nie omijają żadnego kwiatka, który stanie mu na drodze.

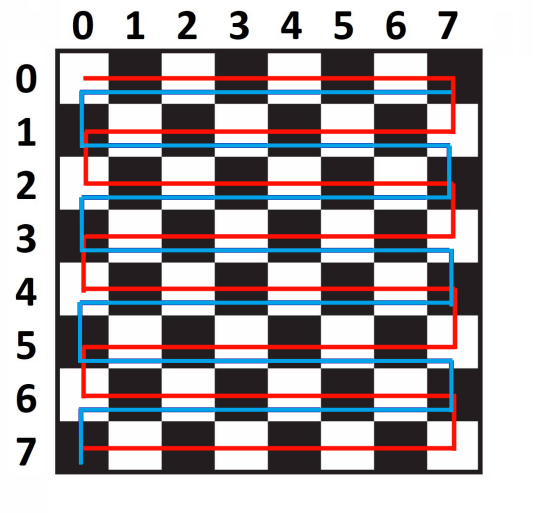
- **TargetingGardener** wprowadza inteligentny sposób poruszania się ogrodnika, ponieważ jest wrażliwy na stan ogrodu.

Rodzaje poruszania się:

➢ **Pattern 1** - “po kolei”, czyli ta sama ścieżka

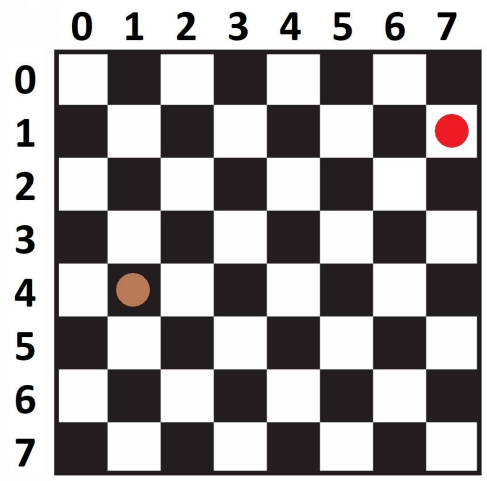


➢ **Pattern 2** - “ósemki”, czyli obróć się na końcu (lustrzane ścieżki)



➢ **Move randomly** - losowo
Ogrodnik idzie w jednym z 8-miu kierunków (uznajemy ruch “na skos”).

➢ **Target the one with the lowest Hp** - targetuj kwiatek o najmniejszym Hp (do którego ogrodnik zdąży dojść przed jego śmiercią - jeśli jest ich kilka to wybierz pierwszy w ogrodzie)

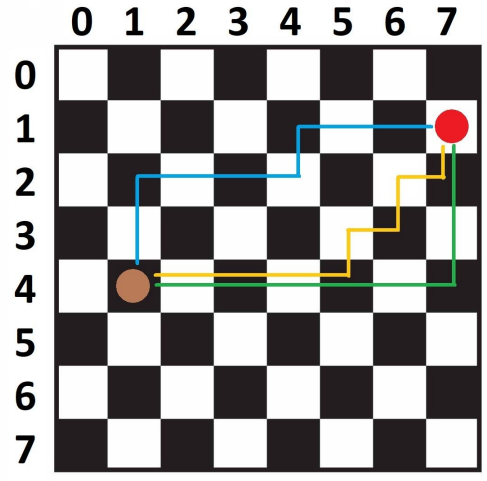


Na początek wyjaśnijmy jakie długości mają różne ścieżki. Załóżmy że mamy taką sytuację:

Brązowa kropka - ogrodnik
Czerwona kropka - kwiatek

Jakie są ich pozycje? Ogrodnik jest na pozycji [1, 4], kwiatek na pozycji [7, 1].
Zastanówmy się, jaka byłaby najkrótsza droga (uznajemy że albo możemy dodać lub odjąć 1 do pierwszej współrzędnej lub do drugiej - czyli nie uznajemy chodzenia "na skos").

Przykład dróg:



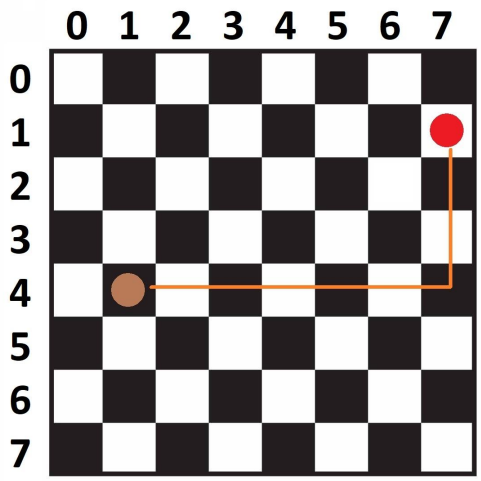
Jaka jest długość drogi **zielonej**? Jest to 9. Jest to przykład bardzo prostej drogi, a jej długość możemy policzyć przez odjęcie odjęcie wektorów (pozycji) naszych obiektów, więc |7-1| + |1-4| = 9.

Policzmy, ile wynosi długość drogi **żółtej** i **niebieskiej**. Jest to również 9.

Zatem nieważne jaką drogę narysujemy (ale ograniczając się do tego aby się zbliżać do celu, a nie oddalać), długość drogi jest taka sama.

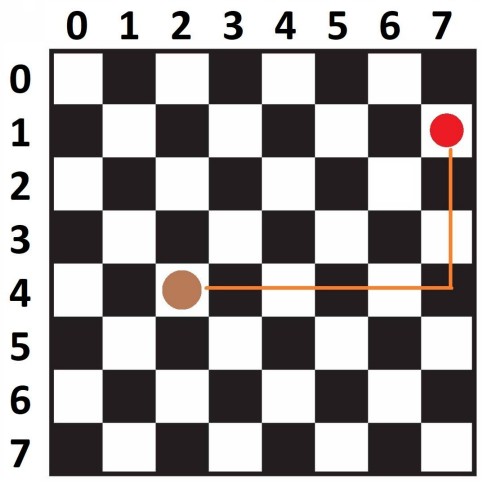
Mając na względzie te zauważenia, TargetingGardener porusza się drogą **żółtą**. Wybieramy ją dlatego, bo ta droga wygląda "lepiej" niż podstawowa droga **zielona** i jest dosyć łatwa do przedstawienia w algorytmie. Jak w takim razie działa ten algorytm? Wyjaśnijmy metodę *moveToTheTarget*.

Idea jest taka, aby ruszyć się w tym kierunku (poziomo lub pionowo) gdzie jesteśmy dalej. Pokażmy to na przykładzie. Na początku mamy tą samą sytuację.



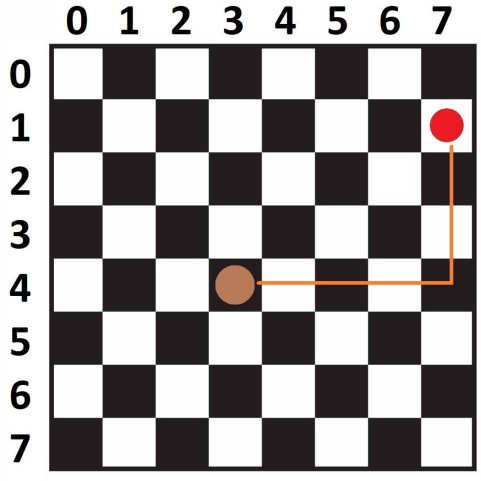
Poziomo jest różnica 6 w pozycjach, pionowo 3. Wybieramy ruch w kierunku takim, gdzie jesteśmy “dalej” - w tym przypadku w poziomo, w prawo.

(1) **Kwiatek** = [7, 1]
Ogrodnik = [1, 4]
distance = [6, -3]
directionVector = [1, 0]



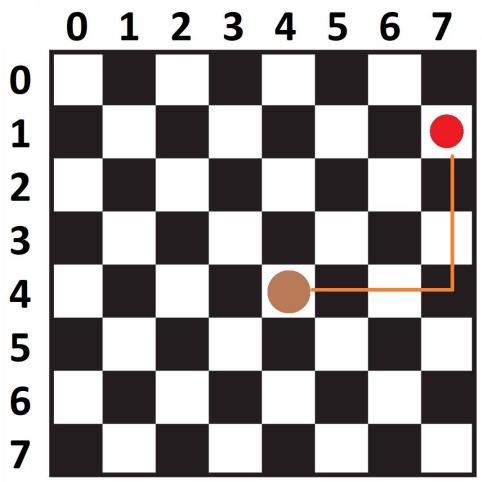
Różnica 5 w poziomie, 3 w pionie - idziemy w poziomie w prawo.

(2) **Kwiatek** = [7, 1]
Ogrodnik = [2, 4]
distance = [5, -3]
directionVector = [1, 0]



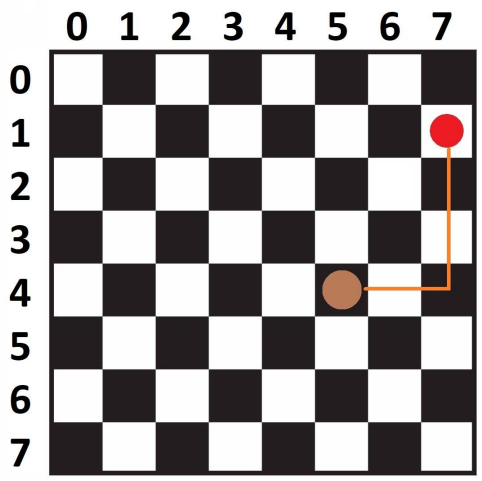
Różnica 4 w poziomie, 3 w pionie - idziemy w poziomie w prawo.

(3) **Kwiatek** = [7, 1]
Ogrodnik = [3, 4]
distance = [4, -3]
directionVector = [1, 0]



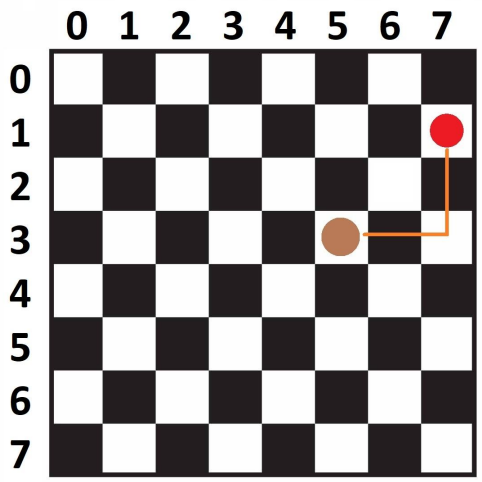
Różnica 3 w poziomie, 3 w pionie - przypadek gdzie te odległości są równe, więc możemy wybrać cokolwiek - idziemy w poziomie w prawo.

(4) **Kwiatek** = [7, 1]
Ogrodnik = [4, 4]
distance = [3, -3]
directionVector = [1, 0]



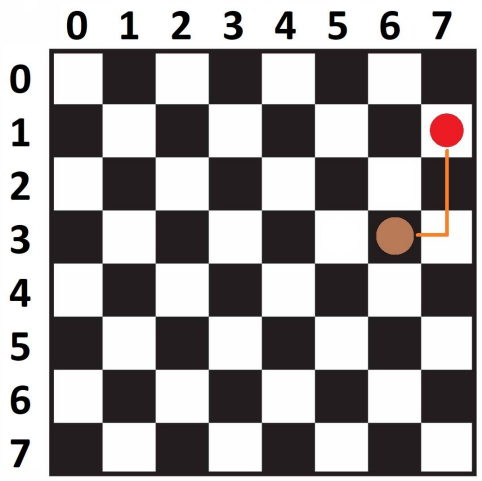
Różnica 2 w poziomie, 3 w pionie - idziemy w pionie w górę.

(5) **Kwiatek** = [7 ,1]
Ogrodnik = [5, 4]
distance = [2, -3]
directionVector = [0, -1]



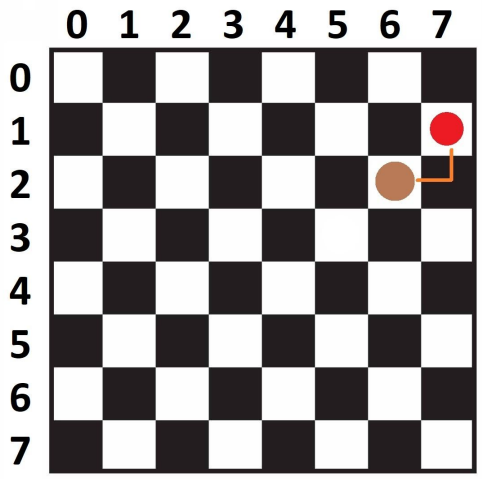
Różnica 2 w poziomie, 2 w pionie - tak jak w poprzednim przypadku równości
- idziemy w poziomie w prawo.

(6) **Kwiatek** = [7 ,1]
Ogrodnik = [5, 3]
distance = [2, -2]
directionVector = [1, 0]



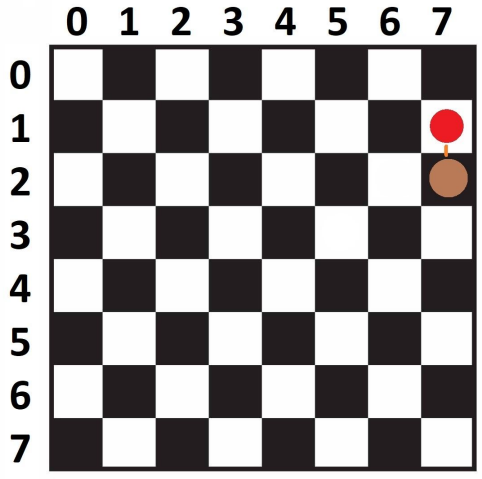
Różnica 1 w poziomie, 2 w pionie - idziemy w pionie w górę.

(7) **Kwiatek** = [7 ,1]
Ogrodnik = [6, 3]
distance = [1, -2]
directionVector = [0, -1]



Różnica 1 w poziomie, 1 w pionie - tak jak w poprzednim przypadku równości
- idziemy w poziomie w prawo.

(8) **Kwiatek** = [7 ,1]
Ogrodnik = [6, 2]
distance = [1, -1]
directionVector = [1, 0]



Różnica 0 w poziomie, 1 w pionie - idziemy w pionie w górę. Trafiliśmy do celu. Droga wyglądała identycznie jak żółta (na przykładzie różnych dróg).

(9) **Kwiatek** = [7 ,1]
Ogrodnik = [7, 2]
distance = [0, -1]
directionVector = [0, -1]

Skoro wiemy jak działa algorytm w praktyce, omówmy działanie kodu.

Załóżmy, że mamy już wybrany kwiatek. Aby ruszyć ogrodnika tworzymy **wektor kierunkowy** ([0,1], [0,-1], [1,0], albo [-1,0]), który dodamy do wektora określającego **pozycję ogrodnika**. Tworzymy go na podstawie pozycji obiektów. Robimy to w ten sposób:

- Tworzymy wektor, który będzie oznaczał o ile pól w jakim kierunku trzeba przesunąć ogrodnika. Dla naszej sytuacji jest to wektor **distance** = [7-1, 1-4] = [6, -3] = [x, y], czyli odejmujemy od pozycji końcowej pozycję startową. Dzięki temu wektorowi wiemy, że musimy poruszyć się o 3 pozycje w górę i 6 w prawo aby dotrzeć do celu.

- Tworzymy dwie zmienne multiplierX i multiplierY, które działają tak, że jeśli pozycja x jest większa od 0, ustaw na 1, wpp. -1. Analogicznie dla y.

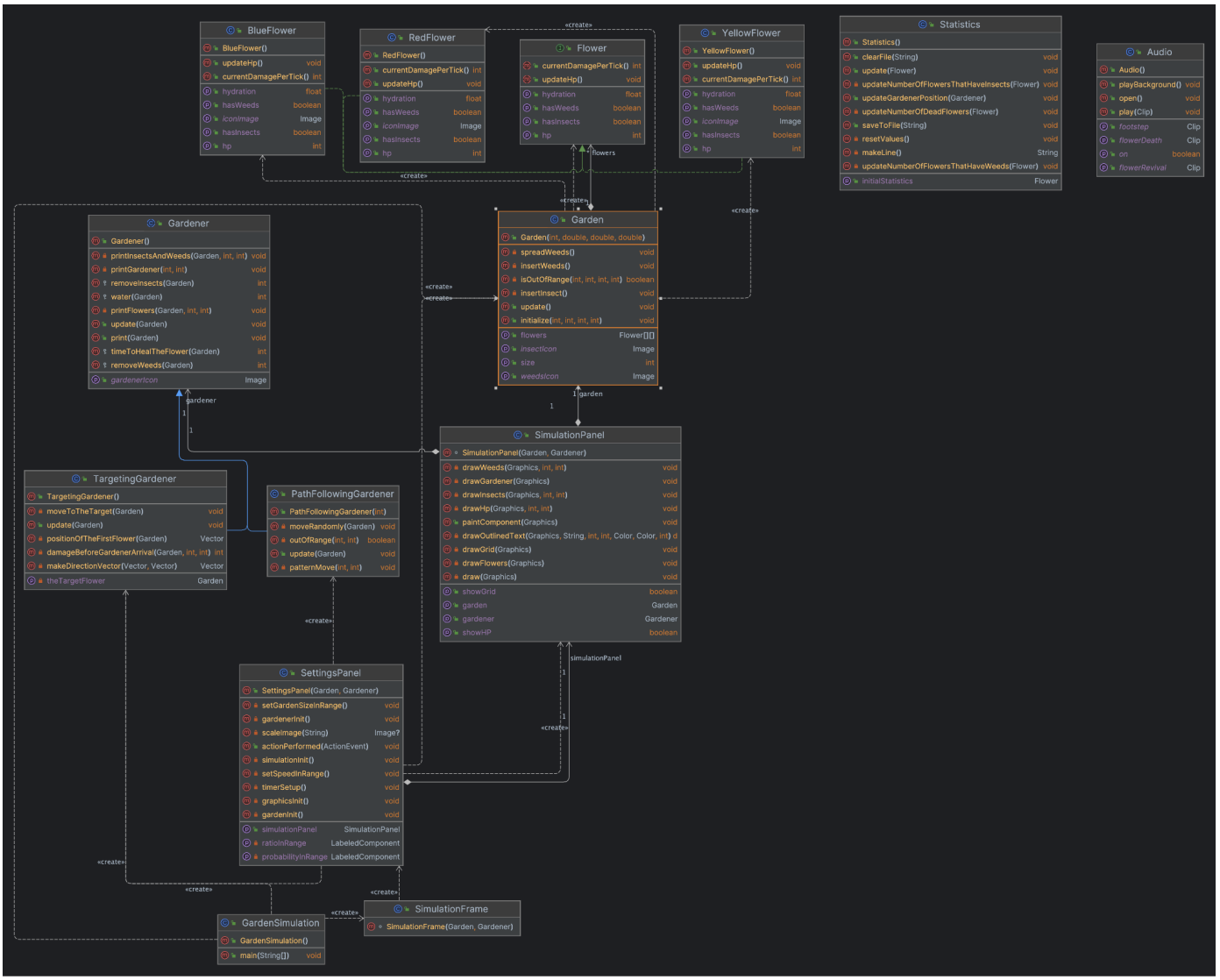
- Współrzędne wektora mnożymy odpowiednio razy multiplierX i multiplierY (otrzymamy wtedy wartość bezwzględną tych współrzędnych czyli [6, 3]).

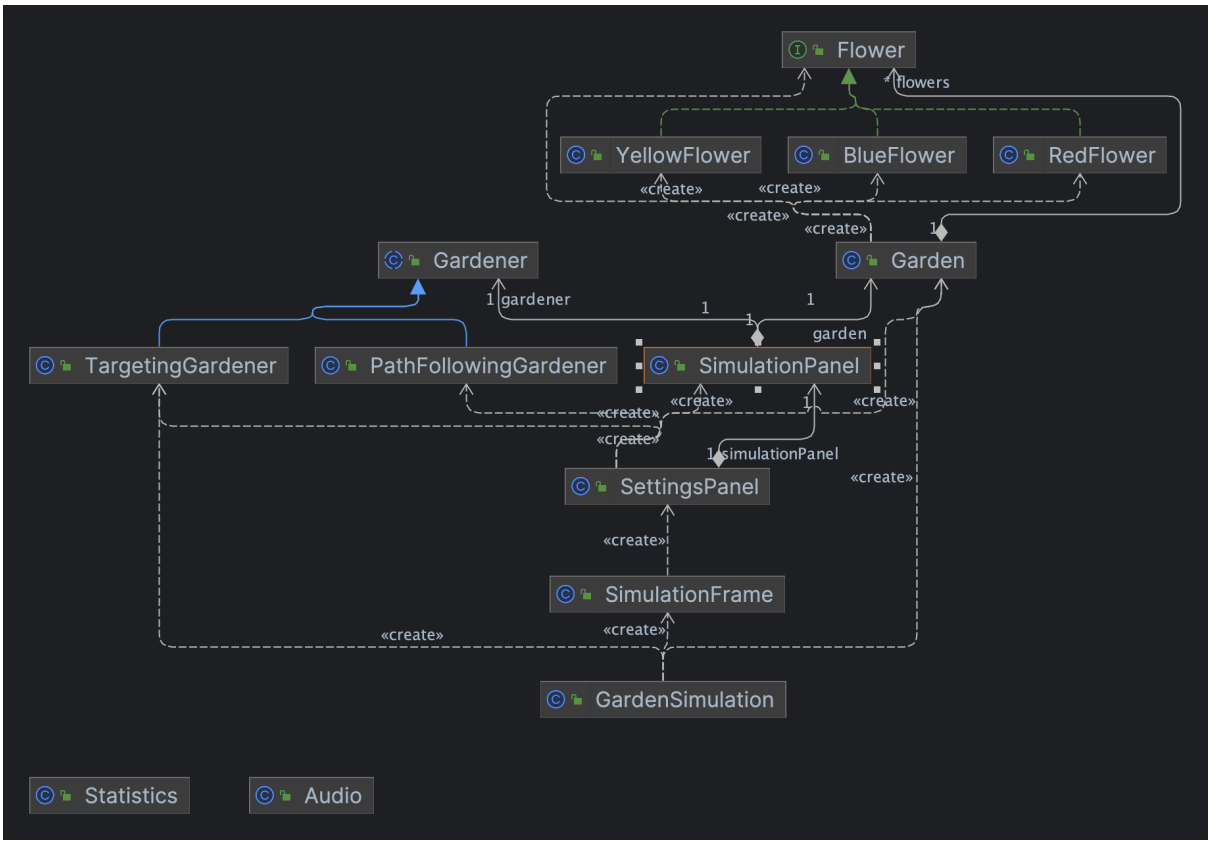
- Jeśli współrzędna x >= współrzędna y, to przesuwamy się w poziomie, czyli tworzymy wektor **directionVector** = [multiplierX, 0]. W przeciwnym przypadku tworzymy wektor [0, multiplierY], aby ruszyć się w pionie. W naszym przypadku na początku stworzyliśmy wektor [1, 0], aby ruszyć się w poziomie w prawo.

- Dodajemy **directionVector** do **pozycji ogrodnika**, aby go ruszyć w tym kierunku. W każdym przypadku przechodzimy przez ten sam proces.

Długość drogi jest nam potrzebna do określenia, czy ogrodnik zdąży dojść do kwiatka zanim umrze. Kiedy ustalamy targetFlower kierujemy się tym, że musi to być kwiatek o najmniejszym HP, do którego ogrodnik może zdążyć
(którego aktualny damage per tick * droga < HP kwiatka).

Diagramy


BlueFlower
BlueFlower()
updateHp() void
currentDamagePerTick() int
hydration float
hasWeeds boolean
iconImage Image
hasInsects boolean
hp int
RedFlower
RedFlower()
currentDamagePerTick() int
updateHp() void
hydration float
hasWeeds boolean
iconImage Image
hasInsects boolean
hp int
Flower
currentDamagePerTick() int
updateHp() void
hydration float
hasWeeds boolean
hasInsects boolean
hp int
YellowFlower
YellowFlower()
updateHp() void
currentDamagePerTick() int
hydration float
hasWeeds boolean
iconImage Image
hasInsects boolean
hp int
Statistics
Statistics()
clearFile(String) void
update(Flower) void
updateNumberOfFlowersThatHaveInsects(Flower) void
updateGardenerPosition(Gardener) void
updateNumberOfDeadFlowers(Flower) void
saveToFile(String) void
resetValues() void
makeLine() String
updateNumberOfFlowersThatHaveWeeds(Flower) void
initialStatistics Flower
Audio
Audio()
playBackground() void
open() void
play(Clip) void
footstep Clip
flowerDeath Clip
on boolean
flowerRevival Clip
Gardener
Gardener()
printInsectsAndWeeds(Garden, int, int) void
printGardener(int, int) void
removeInsects(Garden) int
water(Garden) int
printFlowers(Garden, int, int) void
update(Garden) void
print(Garden) void
timeToHealTheFlower(Garden) int
removeWeeds(Garden) int
gardenerIcon Image
Garden
Garden(int, double, double, double)
spreadWeeds() void
insertWeeds() void
isOutOfRange(int, int, int, int) boolean
insertInsect() void
update() void
initialize(int, int, int, int) void
flowers Flower[][]
insectIcon Image
size int
weedsIcon Image
TargetingGardener
TargetingGardener()
moveToTheTarget(Garden) void
update(Garden) void
positionOfTheFirstFlower(Garden) Vector
damageBeforeGardenerArrival(Garden, int, int) int
makeDirectionVector(Vector, Vector) Vector
theTargetFlower Garden
PathFollowingGardener
PathFollowingGardener(int)
moveRandomly(Garden) void
outOfRange(int, int) boolean
update(Garden) void
patternMove(int, int) void
SimulationPanel
SimulationPanel(Garden, Gardener)
drawWeeds(Graphics, int, int) void
drawGardener(Graphics) void
drawInsects(Graphics, int, int) void
drawHp(Graphics, int, int) void
paintComponent(Graphics) void
drawOutlinedText(Graphics, String, int, int, Color, Color, int) d
drawGrid(Graphics) void
drawFlowers(Graphics) void
draw(Graphics) void
showGrid boolean
garden Garden
gardener Gardener
showHP boolean
SettingsPanel
SettingsPanel(Garden, Gardener)
setGardenSizeInRange() void
gardenerInit() void
scaleImage(String) Image?
actionPerformed(ActionEvent) void
simulationInit() void
setSpeedInRange() void
timerSetup() void
graphicsInit() void
gardenInit() void
simulationPanel SimulationPanel
ratioInRange LabeledComponent
probabilityInRange LabeledComponent
GardenSimulation
GardenSimulation()
main(String[]) void
SimulationFrame
SimulationFrame(Garden, Gardener)
«create»
flowers
gardener
garden
simulationPanel


Flower
*
flowers
YellowFlower
BlueFlower
RedFlower
«create»
«create»
«create»
«create»
1
Gardener
Garden
1
gardener
1
1
1
garden
TargetingGardener
PathFollowingGardener
SimulationPanel
«create»
«create»
1
«create»
«create»
«create»
1
simulationPanel
SettingsPanel
«create»
«create»
SimulationFrame
«create»
«create»
GardenSimulation
Statistics
Audio

Architektura symulacji 

<<Interface>> Flower

Główny twórca: Bartosz Kawiak

**Wyjątki podkreślone*

Pola:

- **int maxHp = 5000** - Maksymalna wartość Hp kwiatka
- **int maxHydration = 100** - Maksymalne nawodnienie kwiatka

Metody:

- **public static Image getIconImage()** - Zwraca teksturę kwiatka
- **public static void setIconImage(Image iconImage)** - Ustawia teksturę kwiatka

- **int getHp()** - Zwraca Hp kwiatka
- **void setHp(int hp)** - Ustawia Hp kwiatka

- **float getHydration()** - Zwraca nawodnienie kwiatka
- **void setHydration(float hydration)** - Ustawia nawodnienie kwiatka

- **boolean getHasInsects()** - Zwraca prawdę lub fałsz - czy kwiatek ma insekta
- **void setHasInsects(boolean hasInsects)** - Ustawia czy kwiatek ma insekta

- **boolean getHasWeeds()** - Zwraca prawdę lub fałsz - czy kwiatek ma chwasty
- **void setHasWeeds(boolean hasWeeds)** - Ustawia czy kwiatek ma chwasty

- **void updateHp()** - Aktualizuje Hp kwiatka w zależności od tego jaki jest jego stan (czy ma insekty itd)
- **int currentDamagePerTick()** - Zwraca ile kwiatek otrzymuje obrażeń przy aktualnym stanie - *Katarzyna Idasz*

<<Class>> RedFlower, YellowFlower, BlueFlower

Implementują interfejs *Flower*

Główny twórca: Bartosz Kawiak

Pola:

- **private int hp** - Wartość Hp kwiatka
- **private float hydration** - Wartość nawodnienia kwiatka
- **private boolean hasInsects** - Czy ma insekta P/F
- **private boolean hasWeeds** - Czy ma chwasty P/F

- **private static Image iconImage** - Tekstura kwiatka

Metody:

- **public RedFlower()/YellowFlower()/BlueFlower()** - Konstruktor - ustawia maksymalne Hp, maksymalne nawodnienie, i fałsz dla posiadania insektów i chwastów
- **+ te opisane w interfejsie *Flower***

<<Class>> Garden

Główny twórca: Bartosz Kawiak

Pola:

- **private final int size** - rozmiar ogrodu
- **private final Flower[][] flowers** - tablica przechowująca kwiaty

- **private final double probabilityOfInsectAppearance** - prawdopodobieństwo pojawienia się insekta (0.0-1.0)
- **private final double probabilityOfWeedsAppearance** - prawdopodobieństwo pojawienia się chwasta (0.0-1.0)
- **private final double probabilityOfWeedsSpread** - prawdopodobieństwo rozprzestrzenienia się chwastów (0.0-1.0)

- **private static Image weedsIcon** - Tesktura chwastów
- **private static Image insectIcon** - Tekstura insektów

Metody:

- **public static void setInsectIcon(Image insectIcon)** - Ustawia teksturę insektów
- **public static Image getInsectIcon()** - Zwraca teksturę insektów

- **public static void setWeedsIcon(Image weedsIcon)** - Ustawia teksturę chwastów
- **public static Image getWeedsIcon()** - Zwraca teksturę chwastów

- **public Garden(int size, double probabilityOfInsectAppearance, double probabilityOfWeedsAppearance, double probabilityOfWeedsSpread)** - Konstruktor - ustawia odpowiedni rozmiar, prawdopodobieństwo pojawienia się insektów, chwastów, rozprzestrzeniania się chwastów.
- **public int getSize()** - Zwraca rozmiar ogrodu
- **public Flower[][] getFlowers()** - Zwraca tablicę kwiatków

- **public void initialize(int redFlowerRatio, int yellowFlowerRatio, int blueFlowerRatio, int emptySpaceRatio)** - Inicjuje ogród na podstawie podanych stosunków rodzajów kwiatków. Rozmieszczenie jest losowe
- **public void update()** - Aktualizuje stan ogrodu (kwiatków) zmniejszając poziom nawodnienia
- **private void insertInsect()** - Ustawia insekta na losowym kwiatku w zależności od prawdopodobieństwa
- **private void insertWeeds()** - Ustawia chwasty na losowym kwiatku w zależności od prawdopodobieństwa
- **private boolean isOutOfRange(int positionX, int positionY, int maxX, int maxY)** - Zwraca prawdę lub fałsz w zależności od tego czy podany punkt jest poza granicami
- **private void spreadWeeds()** - Ustawia chwasta w obrębie kwiatka, który już go ma w zalezności od prawdopodobieństwa

<<Abstract Class>> Gardener

Główny twórca: Katarzyna Idasz

**Wyjątki podkreślone*

Pola:

- **protected int positionX** - pozycja X
- **protected int positionY** - pozycja Y
- **protected int actionTimer** - Ilość czasu jaką musi poświęcić ogrodnik na pielęgnację kwiatka, na którym stoi
- **private static Image gardenerIcon** - Tekstura ogrodnika

Metody:

- **public Gardener()** - Konstruktor - ustawia pozycję i czas na 0

- **public static void setGardenerIcon(Image gardnerIcon)** - Ustawia teksturę ogrodnika - *Bartosz Kawiak*
- **public static Image getGardenerIcon()** - Zwraca teksturę ogrodnika - *Bartosz Kawiak*

- **public void update(Garden garden)** - Aktualizuje stan ogrodnika
- **protected int timeToHealTheFlower(Garden garden)** - Zwraca ile czasu ogrodnik musi poświęcić na pielęgnację kwiatka na którym stoi, w zalezności od jego stanu (czy ma insekty itd)
- **protected int water(Garden garden)** - Ustawia maksymalne nawodnienie i Hp kwiatka, zwraca ilość czasu na podlanie
- **protected int removeInsects(Garden garden)** - Usuwa insekta, zwraca ilość czasu na usunięcie insekta
- **protected int removeWeeds(Garden garden)** - Usuwa chwasty, zwraca ilość czasu na usunięcie chwasta

- **public void print(Garden garden)** - Wypisuje w konsoli ogrodnika, kwiatki i ich stan
- **private void printGardener(int x, int y)** - Wypisuje w konsoli ogrodnika jeśli jest na danej pozycji
- **private void printFlowers(Garden garden, int x, int y)** - Wypisuje w konsoli kwiatek na danej pozycji lub puste pole
- **private void printInsectsAndWeeds(Garden garden, int x, int y)** - Wypisuje w konsoli stan insektów i chwastów kwiatka na danej pozycji oraz Hp

<<Class>> PathFollowingGardener

Rozszerza klasę abstrakcyjną *Gardener*

Główny twórca: Katarzyna Idasz

**Wyjątki podkreślone*

Pola:

- **private int horizontalDirection** -> 1 = idź w prawo, -1 = idź w lewo
- **private int verticalDirection** -> 1 = idź w dół, -1 = idź w górę
- **private final int movementIndex** - Numer stylu poruszania się

Metody:

- **public PathFollowingGardener(int movementIndex)** - Konstruktor - odwołuje się do konstruktora *Gardener* i ustawia poruszanie na “Right” i “Forward” oraz numer stylu poruszania się na zadany
- **@Override**
  public void update(Garden garden) - Aktualizuje stan ogrodnika: Jeśli nie skończył zajmować się kwiatkiem to nadal go pielęgnuje, wpp. rusz się odpowiednim ruchem i oblicz czas na pielęgnację tego kwiatka

- **private patternMove(int maxIndex, int pattern)** - Rusz się w stylu: Pattern 1 - “po kolei”, czyli ta sama ścieżka / Pattern 2 - “ósemkowym”, czyli obróć się na końcu (lustrzane ścieżki)
- **private void moveRandomly(Garden garden)** - Rusz się w losową stronę - *Bartosz Kawiak*
- **private boolean outOfRange(int i, int maxIndex)** - Zwraca prawdę lub fałsz w zależności od tego czy podany punkt jest poza zasięgiem - *Bartosz Kawiak*

<<Class>> Targeting Gardener

Rozszerza klasę abstrakcyjną *Gardener*
Główny twórca: Katarzyna Idasz

Pola:

- **private Vector targetFlower** - Pozycja docelowego kwiatka

Metody:

- **public TargetingGardener()** - Konstruktor - odwołuje się do konstruktora *Gardener* i ustawia docelowy kwiatek na swojej pozycji
- **@Override**
  public void update(Garden garden) - Aktualizuje stan ogrodnika: jeśli nie skończył zajmować się kwiatkiem to nadal go pielęgnuje, wpp. rusz się do docelowego kwiatka

- **private void moveToTheTarget(Garden garden)** - Rusza się do docelowego kwiatka. Jeśli jest na docelowym kwiatku, ustaw czas potrzebny na jego pielęgnację i ustaw nowy docelowy kwiatek
- **private void setTheTargetFlower(Garden garden)** - Ustawia nowy docelowy kwiatek - kwiatek o najmiejszym Hp, do którego zdąży ogrodnik

- **private Vector positionOfTheFirstFlower(Garden garden)** - Zwraca pozycję pierwszego kwiatka w ogrodzie - pomocnicza do poprzedniej metody
- **private int damageBeforeGardenerArrival(Garden garden, int x, int y)** - Zwraca ilość obrażeń jaką otrzyma kwiatek na podanej pozycji przed przyjściem ogrodnika na podstawie ich pozycji - pomocnicza do metody poprzednich dwóch metod
- **private Vector makeDirectionVector(Vector start, Vector end)** - Zwraca wektor kierunkowy na podstawie startu i końca

Prywatne klasy:

- private static class **Vector**:
  - *Pola:*
    - **private int x** - Pozycja X
    - **private int y** - Pozycja Y
  - *Metody:*
    - **public Vector(int x, int y)** - Konstruktor - ustawia pozycje x i y na podane
    - **public Vector(Vector vector)** - Konstruktor kopiujący - ustawia pola na takie jak podany wektor
    - **private void add(Vector vector)** - Dodaje do wektora podany wektor

<<Class>> SimulationFrame

Rozszerza klasę *JPanel*

Główny twórca: Katarzyna Idasz

Metody:

- **SimulationFrame(Garden garden, Gardener gardener)** - Konstruktor - Dodaje panele ustawienia oraz symulacji, ustawia tytuł okna, pozycję okna na środek ekranu i podstawowe parametry

<<Class>> SimulationPanel

Rozszerza klasę *JPanel*

Główny twórca: Katarzyna Idasz

**Wyjątki podkreślone*

Pola:

- **static final int SCREEN_SIZE = 750** - Rozmiar panelu

- **static int UNIT_SIZE** - rozmiar jednostki (tekstury)
- **private Garden garden** - Ogród
- **private Gardener gardener** - Ogrodnik
- **private boolean showHP** - Czy wyświetlać Hp P/F
- **private boolean showGrid** - Czy wyświetlać siatkę P/F

Metody:

- **SimulationPanel(Garden garden, Gardener gardener)** - Konstruktor - ustawia podstawowe parametry panelu, takie jak rozimiar i kolor tła (ogrodu), ustawia ogród i ogrodnika na te podane obiekty, ustawia rozmiar jednostki w zależności od rozmiaru panelu i rozmiaru ogrodu, włącza muzykę w tle

- **public Garden getGarden()** - Zwraca ogóród
- **public void setGarden(Garden garden)** - Ustawia ogród na podany
- **public Gardener getGardener()** - Zwraca ogrodnika
- **public void setGardener(Gardener gardener)** - Ustawia ogrodnika na podanego
- **public void setShowHP(boolean showHP)** - Ustawia wyświetlanie Hp na podane P/F
- **public void setShowGrid(boolean showGrid)** - Ustawia wyświetlanie siatki na podane P/F

- **public void paintComponent(Graphics g)** - Maluje panel
- **private void draw(Graphics g)** - Wywołuje wszystkie metody rysowania
- **private void drawGrid(Graphics g)** - Rysuje siatkę
- **private void drawFlowers(Graphics g)** - Rysuje kwiatki i Hp jeśli showHp == true
- **private void drawWeeds(Graphics g, int x, int y)** - Rysuje chwasty jeśli są na podanej pozycji
- **private void drawInsects(Graphics g, int x, int y)** - Rysuje insekty jeśli są na podanej pozycji
- **private void drawOutlinedText(Graphics g, String text, int x, int y, Color textColor, Color outlineColor, int outlineThickness)** - Rysuje tekst z obrysem - *Bartosz Kawiak*
- **private void drawHp(Graphics g, int x, int y)** - Rysuje tekst (Hp kwiatka na podanej pozycji) metodą wyżej
- **private void drawGardener(Graphics g)** - Rysuje ogrodnika

**Tekstury i dodanie tekstur do metod draw - Bartosz Kawiak*

<<Class>> SettingsPanel

Rozszerza klasę *JPanel,* implementuje *ActionListener*

Główny twórca: Katarzyna Idasz

**Wyjątki podkreślone*

Pola:

- **private final SimulationPanel simulationPanel** - Panel symulacji
- **static final int SCREEN_WIDTH = 300** - Szerokość panelu
- **static final int SCREEN_HEIGHT = 750** - Wysokość panelu
- **private boolean running** - Czy symulacja jest włączona P/F
- **private final Timer timer** - Zegar symulacji

- **private final LabeledComponent speed** - Pole prędkości
- **private final LabeledComponent gardenSize** - Pole rozmiaru ogrodu

- **private final LabeledComponent redRatio** - Pole stosunku czerwonych kwiatków
- **private final LabeledComponent yellowRatio** - Pole stosunku żółtych kwiatków
- **private final LabeledComponent blueRatio** - Pole stosunku niebieskich kwiatków
- **private final LabeledComponent emptyRatio** - Pole stosunku pustych pól

- **private final LabeledComponent probabilityOfInsectAppearance** - Pole prawdopodobieństwa pojawienia się insektów
- **private final LabeledComponent probabilityOfWeedsAppearance** - Pole prawdopodobieństwa pojawienia się chwastów
- **private final LabeledComponent probabilityOfWeedsSpread** - Pole prawdopodobieństwa rozprzestrzenienia się chwastów

- **private final LabeledComponent gardenerMovement** - Przycisk stylu poruszania się ogrodnika
- **private final String[] movementText** - Tablica nazw stylów poruszania się ogrodnika
- **private int movementIndex** - Numer stylu poruszania się ogrodnika

- **private final LabeledComponent showHP** - Check box pokazywania Hp kwiatków
- **private final LabeledComponent showGrid** - Check box wyświetlania siatki

- **private final JButton startStopButton** - Przycisk Start/Stop

- **private final LabeledComponent soundEffects** - Check box efektów dźwiękowych

Metody:

- **public SettingsPanel(Garden garden, Gardener gardener)** - Konstruktor - ustawia podstawowe parametry panelu, takie jak rozmiar, kolor tła, układ komponentów. Tworzy nowu panel symulacyjny. Ustawia wszystkie interaktywne części GUI oraz ustawienia symulacji (zegar o odpowiedniej prędkości)
- **public SimulationPanel getSimulationPanel()** - Zwraca panel symulacyjny

- @**Override**
  public void actionPerformed(ActionEvent e) - Metoda wywołująca się w każdym ticku (akcja). Jeśli symulacja jest włączona to aktualizuje stan ogrodu i ogrodnika. Jeśli użytkownik nacisnął przycisk *Stop* to zatrzymaj symulację, jeśli *Start* to zainicjuj zegar, ogród, ogrodnika, panel symulacyjny, tekstury w zależności od podanych przez użytkownika parametrów. Jeśli użytkownik nacisnął przycisk z opcją poruszania się ogrodnika to zmień napis na nim

- **private void timerSetup()** - Inicjuje zegar po uprzednim ustawieniu wartości w zakresie
- **private void gardenInit()** - Inicjuje ogród (rozmiar, proporcja kwiatków, % insektów i chwastów) po uprzednim ustawieniu wszystkich wartości w zakresie
- **private void gardenerInit()** - Inicjuje ogrodnika w zależności od wybranej przez użytkownika metody poruszania
- **private void simulationInit()** - Inicjuje symulację - rozmiar jednostki, czy pokazywać HP kwiatków i siatkę, włącza dźwięk w zależności od wybranych przez użytkownika opcji
- **private Image scaleImage(String filename)** - Zwraca przeskalowany obraz w zależności od podanego rozmiaru ogrodu - *Bartosz Kawiak*
- **private void graphicsInit()** - Inicjuje tekstury (skaluje je w zależności od podanego rozmiaru ogrodu) - *Bartosz Kawiak*

- **private void setSpeedInRange()** - Ustawia prędkość w zakresie
- **private void setGardenSizeInRange()** - Ustawia rozmiar ogrodu w zakresie

- **private void setRatioInRange(LabeledComponent labeledComponent)** - Ustawia proporcje ilości podanego kwiatka w zakresie
- **private void setProbabilityInRange(LabeledComponent labeledComponent)** - Ustawia prawdopodobieństwo podanego wydarzenia w zakresie

Prywatne klasy:

- private static class **LabeledComponent** (rozszerza klasę *JPanel*)
  - *Pola*:
    - **private final JComponent component** - Główny komponent
    - **private final Dimension componentDimension** = new Dimension(60,30) - Rozmiar komponentu
    - **private final Dimension panelDimension** = new Dimension((int) componentDimension.getWidth() + 20, (int) componentDimension.getHeight() + 40) - Rozmiar panelu
  - *Metody*:
    - **public LabeledComponent(String text, JComponent component)** - Konstuktor - Ustawia podstawowe paramerty panelu (rozmiar, kolor tła). Ustawia podpis komponentu oraz parametry komponentu, w zależności od jego rodzaju
    - //**public void resize(int width)** - Zmienia rozmiar obiektu
    - **public JComponent getComponent()** - Zwraca główny komponent

<<Class>> Audio

Główny twórca: Bartosz Kawiak

Restrukturyzacja klasy - Katarzyna Idasz

Pola:

- **private static boolean on** - Czy efekty dźwiękowe włączone P/F
- **private static Clip footstep** - Dźwięk chodznia
- **private static Clip flowerDeath** - Dźwięk śmierci kwiatka
- **private static Clip flowerRevival** - Dźwięk leczenia kwiatka
- **private static Clip background** - Muzyka w tle

Metody:

- **public static Clip getFootstep()** - Zwraca dźwięk chodzenia
- **public static Clip getFlowerDeath()** - Zwraca dźwięk umierania kwiatka
- **public static Clip getFlowerRevival()** - Zwraca dźwięk leczenia kwiatka

- **public static void setOn(boolean on)** - Ustawia włączenie efektów dźwiękowych na podane P/F

- **public static void open()** - Otwiera wszystkie pliki dźwiękowe
- **public static void play(Clip clip)** - Włącza podany efekt dźwiękowy
- **public static void playBackground()** - Włącza muzykę w tle w pętli

Muzyka i efekty dźwiękowe - Bartosz Kawiak

<<Class>> Statistics

Główny twórca: Bartosz Kawiak

Pola:

- **private static int initialNumberOfRedFlowers = 0** - Początkowa ilość czerwonych kwiatów w ogrodzie
- **private static int initialNumberOfYellowFlowers = 0** - Początkowa ilość żółtych kwiatów w ogrodzie
- **private static int initialNumberOfBlueFlowers = 0** - Początkowa ilość niebieskich kwiatów w ogrodzie

- **private static int numberOfDeadRedFlowers = 0** - Ilość zmarłych kwiatów koloru czerwonego
- **private static int numberOfDeadYellowFlowers = 0** - Ilość zmarłych kwiatów koloru żółtego
- **private static int numberOfDeadBlueFlowers = 0** - Ilość zmarłych kwiatów koloru niebieskiego

- **private static int numberOfFlowersThatHaveInsects = 0** - Ilość insektów w ogrodzie
- **private static int numberOfFlowersThatHaveWeeds = 0** - Ilość chwastów w ogrodzie

- **private static int tick = 1** - Generacja, aktualne pokolenie

Metody:

- **public static void setInitialStatistics(Flower flower)** - Ustawia początkowe ilości kwiatów, które zostały zainicjalizowane w ogrodzie

- **private static void updateNumberOfDeadFlowers(Flower flower)** - Aktualizuje zmienne przechowujące informację o ilośći martwych kwiatów
- **private static void updateNumberOfFlowersThatHaveInsects(Flower flower)** - Aktualizuje zmienne przechowujące informację o ilośći insektów
- **private static void updateNumberOfFlowersThatHaveWeeds(Flower flower)** - Aktualizuje zmienne przechowujące informację o ilośći chwastów
- **private static void resetValues()** - Zeruje zmienne przechowujące ilość zmarłych kwiatów, ilość insektów i ilość chwastów
- **public static void update(Flower flower)** - Realizuje metody aktualizujące ilości zmarłych kwiartów

- **private static String makeLine()** - Zwraca stworzony wiersz ze statystykami
- **public static void saveToFile(String filename)** - Zapisuje wiersz do pliku
- **public static void clearFile(String filename)** - Czyści zawartość pliku i umieszcza w nim wiersz z nagłówkami statystyk

<<Class>> GardenSimulation

Twórcy: Bartosz Kawiak i Katarzyna Idasz

Metody:

- **public static void main(String[] args)** - Metoda main odpowiadająca za włączenie symulacji - tworzy obiekty: ogród, ogrodnika, okno symulacji

---

Cały kod jest do wglądu na GitHubie, tak jak historia i wszystkie poprzednie wersje.

GitHub projektu - link