Felhasználói dokumentáció

A játék elindítása után a program bekéri a felhasználótól a pálya méretét. Két 50-nél nagyobb, de 300-nál kisebb számot kell írni. A két szám a szélesség, illetve a magasságot fogják megadni. A számokat, úgy kell beírni, hogy beírja a felhasználó az első számot, majd szóköz, és beírja a másodikat és entert üt.

Itt megkezdődik a játék. A felhasználó következő feladata, hogy megnyomja az egyik nyilat ábrázoló billentyűt, ezzel megadva a kezdő irányt a kígyónak, majd ezután ezekkel a billentyűkkel fog tudni irányt változtatni. Fontos tudni, hogy teljes 180°-os fordulatot nem tud venni a kígyó, tehát az ellenkező irányba csak úgy lehet menni, ha előtte jobbra vagy balra elfordult a kígyó. Tegyük fel, a felhasználó lenyomta a jobbra nyilat, elindult a kígyó jobbra. Ahhoz hogy balra menjen, vagyis az ellenkező irányba, először le kell nyomnia a fel vagy le nyilat, és így egy „nagyobb” ívben tud megfordulni. Játék közben figyelni kell arra, hogy ne menjen neki a kígyó a pálya szegélyeinek, mert akkor meghal, illetve, hogy falnak se menjen neki. A fal, az egy kék színű vonal a pályán, ami véletlenszerű helyen jelenik meg a kör kezdetén. A felhasználónak minél több pontot kell ütközés nélkül összegyűjteni.

Ahhoz, hogy pontokat szerezzen a felhasználó, gyümölcsöket kell ennie. Ebből háromféle van, fehér, zöld és lila. Mind három gyümölcs ad pontokat a felhasználónak, viszont különböző hatásuk is van a kígyóra. A fehér gyümölcs megnöveli a kígyó méretét, a zöld elteleportálja egy véletlenszerű pozícióra, a lila pedig felgyorsítja. Ha elegendő pontot összegyűjt (50, vagyis a kígyó testének a tízszerese) a felhasználó, akkor nyer, ha pedig neki megy valamilyen falnak, vagy szegélynek mielőtt elérné a maximum pontszámot, akkor veszít.

Vereség esetén újra kell futattni a programot új játék megkezdéséhez.

Kigyó

Tervezzen objektummodellt kígyójáték objektumainak leírására! Legyen pálya, kígyó, gyümölcs, amit a kígyó fel tud szedni. Határozza meg az objektumok kapcsolatát és felelősségét!
Demonstrálja a működést külön modulként fordított tesztprogrammal! A játék állását nem kell grafikusan megjeleníteni, elegendő csak karakteresen, a legegyszerűbb formában! A megoldáshoz **ne** használjon STL tárolót!

Pálya

A pálya objektum alkotja a pályát, a felhasználó segítségével meghatározná annak méretét a, vagyis a szegélyeit, amit elérve a kígyó elpusztul. Véletlenszerűen egy fal jelenik meg a pályán a játék elején, nehezítve a játékos dolgát. Véletlenszerűen jelennek meg gyümölcsök is a pályán, amik szintén hátráltathatják, de akár segíthetnek is az előre jutásban.

Kígyó

A kígyó objektum tudna a szélrózsa minden irányában mozogni a (←), (→), (↑), (↓) billentyűkkel, amik más játékoknak megfelelően, a balra, jobbra, fel, le parancsokat hajtanák végre. A kígyó mérete változó lenne, annak megfelelően, hogy hány darab, illetve milyen fajta gyümölcsöt evett meg. A gyümölcsök megevésével a felhasználó pontokat is szerez.

Gyümölcs

A gyümölcsök helye a pályán, illetve az, hogy éppen melyik jelenik meg, véletlenszerű. A gyümölcsökből kapott pontokból ha elegendőt (50) összeszed a felhasználó, akkor nyer. Minden gyümölcs különböző hatással van a kígyóra. A kígyó mérete, pozíciója és sebessége is változhat az alapján, hogy milyen színű finomságot kebelez be. Az

aktuális pontszám a képernyő jobb felső sarkában van feltűntetve. Abban az esetben ha neki megy a szegélynek, falnak, vagy önmagának, akkor veszít.

Játék működése

A játék működését egy tesztprogram demonstrálja. Minden obejktumnak, aminek a pályán helye van, egy pont osztály segítségével van meghatározva a pozíciója. Illetve a kigyó teste. A kígyó a pálya méreteinek megfelelően tud mozogni, teljesen a felhasználóra bízva annak útvonalát. Amíg nem ütközik, vagy nyer a felhasználó, addig nem ér véget a játék.

Fontosabb algoritmusok

Game:

- set_end_game(bool): beállítja a játék végének állapotát (igaz, vagy hamis)
- set_loaded_game(bool): a betöltött játék állapotát állítja be (igaz, akkor egy régebbi játék folytatása zajlik, ha hamis, akkor új játék)
- save(): elmenti a játékállást egy szöveges fájlba. A kígyó méretét, illetve a pontszámot menti el, emellett az éppen pályán lévő falakat, gyümölcsöket. (formátum: pl.: kígyó méret;pontszám;stb.)
- load(): betölti az elmentett játékot.
- quit(): kilép a játékból, és bezárul a program.
- print_field(): kirajzolja a pályát a képernyőre.
- test(): teszteseteket futtató ciklus függvény.
- game_loop(): a játékot futtató függvény.

Field:

- wall_generate(List<Point>&, List<Point>&): random pozíciókban falat rak a pályára. Figyeli, hogy ne oda rakjon, ahol van másik fal, vagy egy gyümölcs.
- fruit_generate(List<Point>&, List<Point>&): gyümölcsöt general.
- snake_generate(Snake&) : a kígyót generálja le.
- get_wall(): lekérdezi, a falakat.
- get_width(): pályaszélességének visszaadása.
- get_height(): pálya magasságának visszaadása.
- set_width(int): szélességet beállító függvény.
- set_height(int): magasságot beállító függvény.

Snake:

- get_size(): kígyó méretét adja vissza.
- get_head(): kigyó fejének pozícióját adja vissza. Illetve van egy konstans verziója is ennek.
- get_body(): a listában tárolt testetrészek pozícióit adja vissza. Illetve konstans változat.
- get_direction(): vissza adja a mozgás irányát szám formájában.
- get_score(): visszaadja a pontszámot.
- get_speed(): visszaadja a kígyó sebességét.
- set_size(int): a kígyó méretét beállítja.
- set_score(int): beállítja a pontszámot.
- set_speed(double): a kígyó sebességét állítja be.

- set_direction(int): a kígyó irányát állítja be.
- move(Field&): a kígyó mozgását végzi az iránynak megfelelően.
- direction(): irányt ad a kígyónak, hogy merre kell mozognia és ez alapján kezd el mozogni. (w, a, s, d billentyűk alapján dönti el)
- crash(const Field&): ellenörzi, hogy volt e ütközés a pályán valahol, akár fallal, akár a pálya határaival.
- eat(Fruit*&): ellenörzi, hogy megegyeznek-e az x,y koordinátái a kígyó fejének a gyümölcsével.
- size_inc(): a kígyó méretét növeli, amikor megeszik egy gyümölcsöt.
- teleport(Field&): a kígyó elteleportálása random pozícióra.
- print(int, int, int): kígyó testrész kirajzolása (fej is), illetve törlése.
- get_tail(): utolsó elem (farok) megkeresése.
- print_score(): kiírja a pontszámot a képernyőre
- speed_up(): a kígyó sebességét növeli.

Fruit:

- get_pos(): vissza adja a gyümölcsök listáját.
- set_pos(int): a gyümölcs pozícióját állítja.
- get_type(): visszaadja a gyümölcs típusát.
- set_type(): beállítja a gyümölcs típusát.
- affect(): a kígyóra tett hatás függvénye az alaposztályban.

White:

- affect(Snake&, List<Point>&, Field&, Game&): a fehér gyümölcs hatása a kígyóra.

Green:

- affect(Snake&, List<Point>&, Field&): a zöld gyümölcs hatása a kígyóra.

Point:

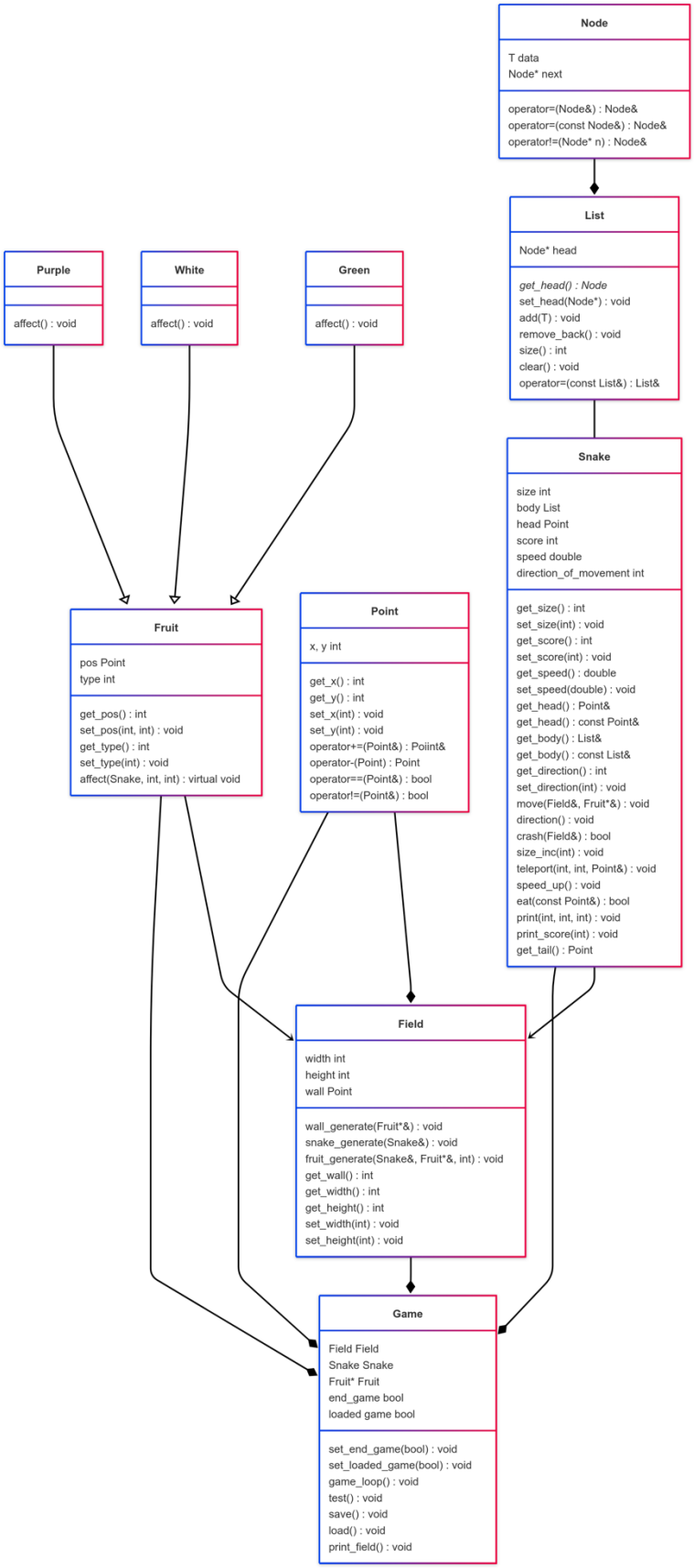
- get_x(): x koordinátát visszaadja.
- get_y(): y koordinátát adja vissza.
- set_x(int): vízszintes koordináta beállítása.
- set_y(int): függőleges koordináta beállítása.
- operator+=(const Point&): hozzá tudunk adni pontot ponthoz.
- operator-(Point&): ki is tudunk vonni pontból pontot.
- operator==(const Point&): egyenlőséget ellenrőrzi.
- operator!=(Point&): különbözőséget ellenörző operátor.

List:

- add(template): egy új elemet ad hozzá a listához.
- remove_back(T): kitörli a lista utolsó elemét.
- clear(): törli a lista összes elemét.
- get_head(): visszaadja a lista fejét.
- set_head(Node<template>*): beállítja a lista fejét.
- size(): a lista méretét határozza meg.
- operator=(const List&): másoló operator.

Node:

- operator=(Node&): egyenlőség operator.
- operator=(const Node&): egyenlőség operator konstans adatokhoz.
- operator!=(Node*): nem egyenlő operator, ellenőrzi, hogy nem különbözőek-e.


Node
T data
Node* next
operator=(Node&) : Node&
operator=(const Node&) : Node&
operator!=(Node* n) : Node&
List
Node* head
get_head() : Node
set_head(Node*) : void
add(T) : void
remove_back() : void
size() : int
clear() : void
operator=(const List&) : List&
Purple
affect() : void
White
affect() : void
Green
affect() : void
Snake
size int
body List
head Point
score int
speed double
direction_of_movement int
get_size() : int
set_size(int) : void
get_score() : int
set_score(int) : void
get_speed() : double
set_speed(double) : void
get_head() : Point&
get_head() : const Point&
get_body() : List&
get_body() : const List&
get_direction() : int
set_direction(int) : void
move(Field&, Fruit*&) : void
direction() : void
crash(Field&) : bool
size_inc(int) : void
teleport(int, int, Point&) : void
speed_up() : void
eat(const Point&) : bool
print(int, int, int) : void
print_score(int) : void
get_tail() : Point
Fruit
pos Point
type int
get_pos() : int
set_pos(int, int) : void
get_type() : int
set_type(int) : void
affect(Snake, int, int) : virtual void
Point
x, y int
get_x() : int
get_y() : int
set_x(int) : void
set_y(int) : void
operator+=(Point&) : Poiint&
operator-(Point) : Point
operator==(Point&) : bool
operator!=(Point&) : bool
Field
width int
height int
wall Point
wall_generate(Fruit*&) : void
snake_generate(Snake&) : void
fruit_generate(Snake&, Fruit*&, int) : void
get_wall() : int
get_width() : int
get_height() : int
set_width(int) : void
set_height(int) : void
Game
Field Field
Snake Snake
Fruit* Fruit
end_game bool
loaded game bool
set_end_game(bool) : void
set_loaded_game(bool) : void
game_loop() : void
test() : void
save() : void
load() : void
print_field() : void