Міністерство освіти і науки України

Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Програмування інтелектуальних інформаційних систем

**ЗВІТ**

до лабораторних робіт

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Виконав**  **студент** |  | ІТ-92 Щур А. С. |  |  |
|  |  | (№ групи, прізвище, ім’я, по батькові ) |  |  |
|  |  |  |  |  |
| **Прийняв** |  | пос. Баришич Л. М. |  |  |
|  |  | (посада, прізвище, ім’я, по батькові ) |  |  |

Київ 2021

# **Завдання лабораторної роботи 8**

# Ця робота призначена для того, щоб ви могли проявити уяву. В гру потрібно додати "Режим Бога". Для цього, втгру потрібно додати інтерактивну консоль, що приймає на себе прості команди, і дозволяє впливати на процеси гри. Ви можете змінювати люі параметри - алгоритми, гравітацію, швидкість тощо - бажано використовувати ті компоненти, що ви застосовували у попередніх роботах. Система вводу команд має бути подібною до макрос-мови і мати прості правила синтаксису. Додайте мінімум 7 різних команд у вашу консоль.

# **Опис програмного коду**

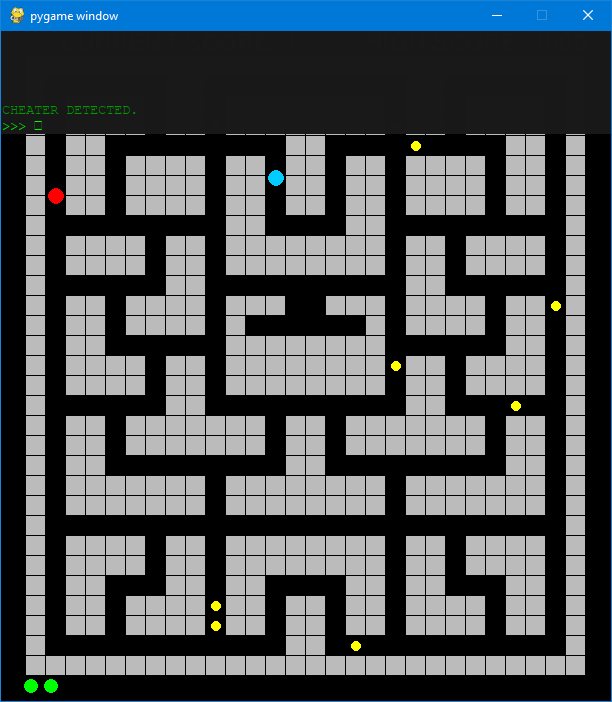
Код викликів консолі:

class App:  
 def \_\_init\_\_(self):  
 self.screen = pygame.display.set\_mode((WIDTH, HEIGHT))  
 self.background = None  
 self.clock = pygame.time.Clock()  
 self.running = True  
 self.state = MENU  
 self.cell\_width = MAZE\_WIDTH//COLS  
 self.cell\_height = MAZE\_HEIGHT//ROWS  
 self.walls = []  
 self.coins = []  
 self.teleports = []  
 self.enemies = []  
 self.e\_pos = []  
 self.p\_pos = (1, 1)  
 self.map\_generator = Generator()  
 self.grid\_map = None  
 self.load\_map()  
 self.player = Player(self, vec(self.p\_pos))  
 self.high\_score = self.load\_score()  
 self.start\_time = time.time()  
 self.wall\_color = GREY  
 self.console = PyCon.PyCon(self.screen,  
 (0, 0, MAZE\_WIDTH + PADDING \* 2, MAZE\_HEIGHT / 6),  
 functions={"map\_change\_color": self.change\_color,  
 "bot\_change\_algo": self.change\_bot\_algo,  
 "bot\_stop": self.stop\_bot,  
 "player\_god": self.god,  
 "player\_can\_eat\_enemies": self.eat\_enemies,  
 "player\_set\_score\_for\_coin": self.set\_score\_for\_coin,  
 "player\_set\_score": self.set\_score,  
 "bot\_add": self.add\_bot\_to\_field,  
 },  
 key\_calls={},  
 vari={"A": 100, "B": 200, "C": 300},  
 syntax={re\_function: console\_func})  
  
  
 def add\_bot\_to\_field(self, value):  
 if 4 >= value > 0:  
 for \_ in range(value):  
 empty\_positions = self.map\_generator.get\_empty\_blocks\_positions()  
 position = random.choice(empty\_positions)  
 self.enemies.append(Enemy(self, vec(position), RANDOM))  
  
 def set\_score(self, value):  
 self.player.current\_score = value  
  
 def set\_score\_for\_coin(self, value):  
 if 10000 >= value >= 0:  
 self.player.score\_for\_coin = value  
  
 def eat\_enemies(self, can\_eat):  
 if can\_eat == 1:  
 self.player.can\_eat\_enemy = True  
 else:  
 self.player.can\_eat\_enemy = False  
  
 def god(self, is\_god):  
 if is\_god == 1:  
 self.player.god = True  
 else:  
 self.player.god = False  
  
 def stop\_bot(self, \*args):  
 if len(args) == 0:  
 for enemy in self.enemies:  
 enemy.personality = STOP  
 else:  
 if len(self.enemies) >= args[0] > 0:  
 self.enemies[args[0] - 1].personality = STOP  
  
 def change\_color(self, color):  
 self.wall\_color = color  
  
 def change\_bot\_algo(self, bot\_number, algo):  
 if len(self.enemies) >= bot\_number > 0:  
 if algo in [RANDOM, DEFAULT]:  
 self.enemies[bot\_number - 1].personality = algo

Код самої консолі, що працює як парсер тексту і приймає першим аргументом команду, іншим аргументи, що передаються в функцію.

import pygame  
from string import ascii\_letters  
import textwrap  
import re  
  
  
re\_token = re.compile(r"""[\"].\*?[\"]|[\{].\*?[\}]|[\(].\*?[\)]|[\[].\*?[\]]|\S+""")  
re\_is\_list = re.compile(r'^[{\[(]')  
re\_is\_number = re.compile(r"""  
(?x)  
[-]?[0][x][0**-**9a**-**fA**-**F]+[lLjJ]? | # Hexadecimal  
[-]?[0][0**-**7]+[lLjJ]? | # Octal  
[-]?[\d]+(?:[.][\d]\*)?[lLjJ]? # Decimal (Int or float)  
""")  
re\_is\_assign = re.compile(r'[$](?P<name>[a**-**zA**-**Z\_]+\S\*)\s\*[=]\s\*(?P<value>.+)')  
re\_is\_comment = re.compile(r'\s\*#.\*')  
re\_is\_var = re.compile(r'^[$][a**-**zA**-**Z\_]+\w\*\Z')  
  
  
class PyCon:  
 def \_\_init\_\_(self, screen, rect, functions=None, key\_calls=None, vari=None, syntax=None):  
 self.message\_of\_the\_day = ["CHEATER DETECTED."]  
  
 self.bg\_color = (25, 25, 25)  
 self.bg\_alpha = 150 # How transparent is the Console Background  
 self.txt\_color\_i = (0, 255, 0) # Green bright Input Color  
 self.txt\_color\_o = (0, 153, 0) # Dark Green output Color  
  
 self.changed = True  
 self.active = False  
 self.preserve\_events = True  
 self.repeat\_rate = [500, 30]  
  
 self.ps1 = "] "  
 self.ps2 = ">>> "  
 self.ps3 = "... "  
 self.c\_ps = self.ps2  
  
 self.c\_out = self.message\_of\_the\_day  
 self.c\_in = ""  
 self.c\_hist = [""]  
 self.c\_hist\_pos = 0  
 self.c\_pos = 0  
 self.c\_draw\_pos = 0  
 self.c\_scroll = 0  
  
 self.parent\_screen = screen  
 self.rect = pygame.Rect(rect)  
 self.size = self.rect.size  
  
 self.font = pygame.font.SysFont("Courier New", 14)  
 self.font\_height = self.font.get\_linesize()  
 self.max\_lines = int((self.size[1] / self.font\_height) - 1)  
 self.max\_chars = int(((self.size[0]) / (self.font.size(ascii\_letters)[0]/len(ascii\_letters))) - 1)  
 self.txt\_wrapper = textwrap.TextWrapper()  
  
 self.bg\_layer = pygame.Surface(self.size)  
 self.bg\_layer.set\_alpha(self.bg\_alpha)  
 self.txt\_layer = pygame.Surface(self.size)  
 self.txt\_layer.set\_colorkey(self.bg\_color)  
  
 pygame.key.set\_repeat(\*self.repeat\_rate)  
  
 self.key\_calls = {}  
 self.func\_calls = {}  
 self.user\_vars = vari  
 self.user\_syntax = syntax  
 self.user\_namespace = {}  
  
 self.add\_key\_calls({"l": self.clear, "c": self.clear\_input, "w": self.set\_active})  
 self.add\_key\_calls(key\_calls)  
  
 self.add\_functions\_calls({"help": self.help, "echo": self.output, "clear": self.clear})  
 self.add\_functions\_calls(functions)  
  
 def screen(self):  
 return self.parent\_screen  
  
 def add\_functions\_calls(self, functions):  
 if isinstance(functions, dict):  
 self.func\_calls.update(functions)  
  
 def add\_key\_calls(self, functions):  
 if isinstance(functions, dict):  
 self.key\_calls.update(functions)  
  
 def output(self, text):  
 *"""Print a string on the Console. Use: echo "Test Test Test" """* self.changed = True  
 if not isinstance(text, str):  
 text = str(text)  
 text = text.expandtabs()  
 text = text.splitlines()  
 self.txt\_wrapper.width = self.max\_chars  
 for line in text:  
 for w in self.txt\_wrapper.wrap(line):  
 self.c\_out.append(w)  
  
 def submit\_input(self, text):  
 self.clear\_input()  
 self.output(self.c\_ps + text)  
 self.c\_scroll = 0  
 self.send\_pyconsole(text)  
  
 def format\_input\_line(self):  
 text = self.c\_in[:self.c\_pos] + "\v" + self.c\_in[self.c\_pos + 1:]  
 n\_max = int(self.max\_chars - len(self.c\_ps))  
 vis\_range = self.c\_draw\_pos, self.c\_draw\_pos + n\_max  
 return self.c\_ps + text[vis\_range[0]:vis\_range[1]]  
  
 def str\_insert(self, text, strn):  
 string = text[:self.c\_pos] + strn + text[self.c\_pos:]  
 self.set\_pos(self.c\_pos + len(strn))  
 return string  
  
 def clear\_input(self):  
 self.c\_in = ""  
 self.c\_pos = 0  
 self.c\_draw\_pos = 0  
  
 def set\_pos(self, newpos):  
 self.c\_pos = newpos  
 if (self.c\_pos - self.c\_draw\_pos) >= int(self.max\_chars - len(self.c\_ps)):  
 self.c\_draw\_pos = max(0, int(self.c\_pos - (self.max\_chars - len(self.c\_ps))))  
 elif self.c\_draw\_pos > self.c\_pos:  
 self.c\_draw\_pos = self.c\_pos - (self.max\_chars/2)  
 if self.c\_draw\_pos < 0:  
 self.c\_draw\_pos = 0  
 self.c\_pos = 0  
  
 def set\_active(self, b=None):  
 if not b:  
 self.active = not self.active  
 else:  
 self.active = b  
 print("CHEEEEAAAAATEEEER")  
  
 def add\_to\_history(self, text):  
 self.c\_hist.insert(-1, text)  
 self.c\_hist\_pos = len(self.c\_hist) - 1  
  
 def draw(self):  
 if not self.active:  
 return  
  
 if self.changed:  
 self.changed = False  
 self.txt\_layer.fill(self.bg\_color)  
 lines = self.c\_out[-(self.max\_lines + self.c\_scroll):len(self.c\_out) - self.c\_scroll]  
 y\_pos = self.size[1]-(self.font\_height\*(len(lines)+1))  
  
 for line in lines:  
 tmp\_surf = self.font.render(line, True, self.txt\_color\_o)  
 self.txt\_layer.blit(tmp\_surf, (1, y\_pos, 0, 0))  
 y\_pos += self.font\_height  
  
 tmp\_surf = self.font.render(self.format\_input\_line(), True, self.txt\_color\_i)  
 self.txt\_layer.blit(tmp\_surf, (1, self.size[1]-self.font\_height, 0, 0))  
  
 self.bg\_layer.fill(self.bg\_color)  
 self.bg\_layer.blit(self.txt\_layer, (0, 0, 0, 0))  
  
 self.parent\_screen.blit(self.bg\_layer, self.rect)  
  
 def process\_input(self, eventlist):  
 if not self.active:  
 return  
  
 for event in eventlist:  
 if event.type == pygame.KEYDOWN:  
 self.changed = True  
 # Special Character Manipulation  
 if event.key == pygame.K\_TAB:  
 self.c\_in = self.str\_insert(self.c\_in, " ")  
 elif event.key == pygame.K\_BACKSPACE:  
 if self.c\_pos > 0:  
 self.c\_in = self.c\_in[:self.c\_pos - 1] + self.c\_in[self.c\_pos:]  
 self.set\_pos(self.c\_pos - 1)  
 elif event.key == pygame.K\_DELETE:  
 if self.c\_pos < len(self.c\_in):  
 self.c\_in = self.c\_in[:self.c\_pos] + self.c\_in[self.c\_pos + 1:]  
 elif event.key == pygame.K\_RETURN or event.key == 271:  
 self.submit\_input(self.c\_in)  
 # Changing Cursor Position  
 elif event.key == pygame.K\_LEFT:  
 if self.c\_pos > 0:  
 self.set\_pos(self.c\_pos - 1)  
 elif event.key == pygame.K\_RIGHT:  
 if self.c\_pos < len(self.c\_in):  
 self.set\_pos(self.c\_pos + 1)  
 elif event.key == pygame.K\_HOME:  
 self.set\_pos(0)  
 elif event.key == pygame.K\_END:  
 self.set\_pos(len(self.c\_in))  
 # History Navigation  
 elif event.key == pygame.K\_UP:  
 if len(self.c\_out):  
 if self.c\_hist\_pos > 0:  
 self.c\_hist\_pos -= 1  
 self.c\_in = self.c\_hist[self.c\_hist\_pos]  
 self.set\_pos(len(self.c\_in))  
 elif event.key == pygame.K\_DOWN:  
 if len(self.c\_out):  
 if self.c\_hist\_pos < len(self.c\_hist) - 1:  
 self.c\_hist\_pos += 1  
 self.c\_in = self.c\_hist[self.c\_hist\_pos]  
 self.set\_pos(len(self.c\_in))  
 # Scrolling  
 elif event.key == pygame.K\_PAGEUP:  
 if self.c\_scroll < len(self.c\_out) - 1:  
 self.c\_scroll += 1  
 elif event.key == pygame.K\_PAGEDOWN:  
 if self.c\_scroll > 0:  
 self.c\_scroll -= 1  
 # Normal character printing  
 elif event.key >= 32:  
 mods = pygame.key.get\_mods()  
 if mods & pygame.KMOD\_CTRL:  
 if event.key in range(256) and chr(event.key) in self.key\_calls:  
 self.key\_calls[chr(event.key)]()  
 else:  
 char = str(event.unicode)  
 self.c\_in = self.str\_insert(self.c\_in, char)  
  
 def send\_pyconsole(self, text):  
 if not text: # Output a blank row if nothing is entered  
 self.output("")  
 return  
  
 self.add\_to\_history(text)  
  
 # Determine if the statement is an assignment  
 assign = re\_is\_assign.match(text)  
  
 # If it is tokenize only the "value" part of $name = value  
 if assign:  
 tokens = self.tokenize(assign.group('value'))  
 else:  
 tokens = self.tokenize(text)  
  
 if tokens is None:  
 return  
  
 # Evaluate  
 try:  
 out = None  
 # A variable alone on a line  
 if (len(tokens) == 1) and re\_is\_var.match(text) and not assign:  
 out = tokens[0]  
 # Statement in the form $name = value  
 elif (len(tokens) == 1) and assign:  
 self.setvar(assign.group('name'), tokens[0])  
 else:  
 # Function  
 out = self.func\_calls[tokens[0]](\*tokens[1:])  
 # Assignment from function's return value  
 if assign:  
 self.setvar(assign.group('name'), out)  
  
 if out is not None:  
 self.output(out)  
 except (KeyError, TypeError):  
 self.output("Unknown Command: " + str(tokens[0]))  
 self.output(r'Type "help" for a list of commands.')  
  
 def setvar(self, name, value):  
 *"""Sets the value of a variable"""* if name in self.user\_vars or name not in self.\_\_dict\_\_:  
 self.user\_vars.update({name: value})  
 self.user\_namespace.update(self.user\_vars)  
 elif name in self.\_\_dict\_\_:  
 self.\_\_dict\_\_.update({name: value})  
  
 def convert\_token(self, tok):  
 tok = tok.strip("$")  
 try:  
 tmp = eval(tok, self.\_\_dict\_\_, self.user\_namespace)  
 except SyntaxError as strerror:  
 self.output("SyntaxError: " + str(strerror))  
 raise ParseError(tok)  
 except TypeError as strerror:  
 self.output("TypeError: " + str(strerror))  
 raise ParseError(tok)  
 except NameError as strerror:  
 self.output("NameError: " + str(strerror))  
 except:  
 self.output("Error:")  
 raise ParseError(tok)  
 else:  
 return tmp  
  
 def tokenize(self, s):  
 if re\_is\_comment.match(s):  
 return [s]  
  
 for r in self.user\_syntax:  
 group = r.match(s)  
 if group:  
 self.user\_syntax[r](self, group)  
 return  
  
 tokens = re\_token.findall(s)  
 tokens = [i.strip("\"") for i in tokens]  
 cmd = []  
 i = 0  
 while i < len(tokens):  
 t\_count = 0  
 val = tokens[i]  
  
 if re\_is\_number.match(val):  
 cmd.append(self.convert\_token(val))  
 elif re\_is\_var.match(val):  
 cmd.append(self.convert\_token(val))  
 elif val == "True":  
 cmd.append(True)  
 elif val == "False":  
 cmd.append(False)  
 elif re\_is\_list.match(val):  
 while not balanced(val) and (i + t\_count) < len(tokens) - 1:  
 t\_count += 1  
 val += tokens[i + t\_count]  
 else:  
 if (i + t\_count) < len(tokens):  
 cmd.append(self.convert\_token(val))  
 else:  
 raise ParseError(val)  
 else:  
 cmd.append(val)  
 i += t\_count + 1  
 return cmd  
  
 def clear(self):  
 *"""Clear the screen! Use: clear"""* self.c\_out = ["[Screen Cleared]"]  
 self.c\_scroll = 0  
  
 def help(self, \*args):  
 if args:  
 items = [(i, self.func\_calls[i]) for i in args if i in self.func\_calls]  
 for i, v in items:  
 out = i + ": Takes %d arguments. " % (v.\_\_code\_\_.co\_argcount - (v.\_\_code\_\_.co\_varnames[0] == "self"))  
 doc = v.\_\_doc\_\_  
 if doc:  
 out += textwrap.dedent(doc)  
 tmp\_indent = self.txt\_wrapper.subsequent\_indent  
 self.txt\_wrapper.subsequent\_indent = " " \* (len(i) + 2)  
 self.output(out)  
 self.txt\_wrapper.subsequent\_indent = tmp\_indent  
 else:  
 out = "Available commands: " + str(self.func\_calls.keys()).strip("[]")  
 self.output(out)  
 self.output(r'Type "help command-name" for more information on that command')  
  
 def write\_history\_to\_file(self):  
 hist\_file\_text = open("History\_text.txt", "w")  
 for item\_h in self.c\_out:  
 hist\_file\_text.write("%s\n" % item\_h)  
 hist\_file\_text.close()  
  
 hist\_file\_cmd = open("History\_cmd.txt", "w")  
 for item\_h in self.c\_hist:  
 hist\_file\_cmd.write("%s\n" % item\_h)  
 hist\_file\_cmd.close()  
  
  
class ParseError(Exception):  
 def \_\_init\_\_(self, token):  
 self.token = token  
  
 def at\_token(self):  
 return self.token  
  
  
def balanced(t):  
 stack = []  
 pairs = {"\'": "\'", '\"': '\"', "{": "}", "[": "]", "(": ")"}  
 for char in t:  
 if stack and char == pairs[stack[-1]]:  
 stack.pop()  
 elif char in pairs:  
 stack.append(char)  
 return not bool(stack)

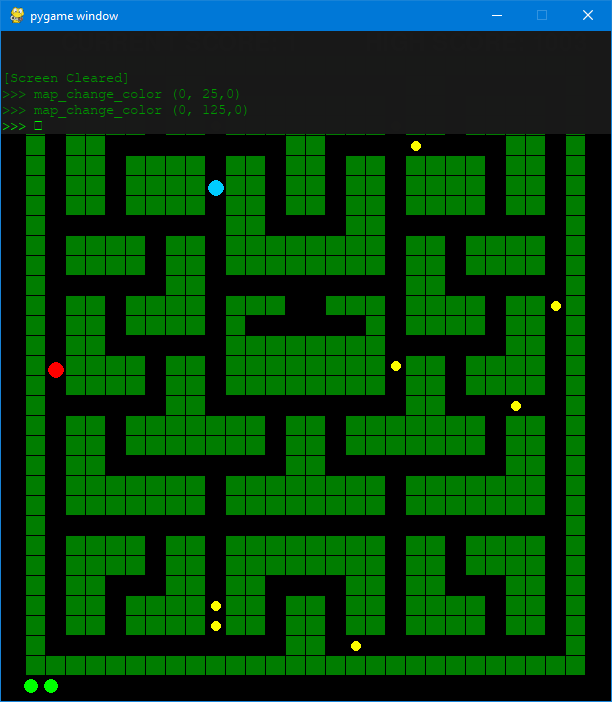
Було розроблено консоль, що дозволяє вводити команди, сама консоль виглядає так:



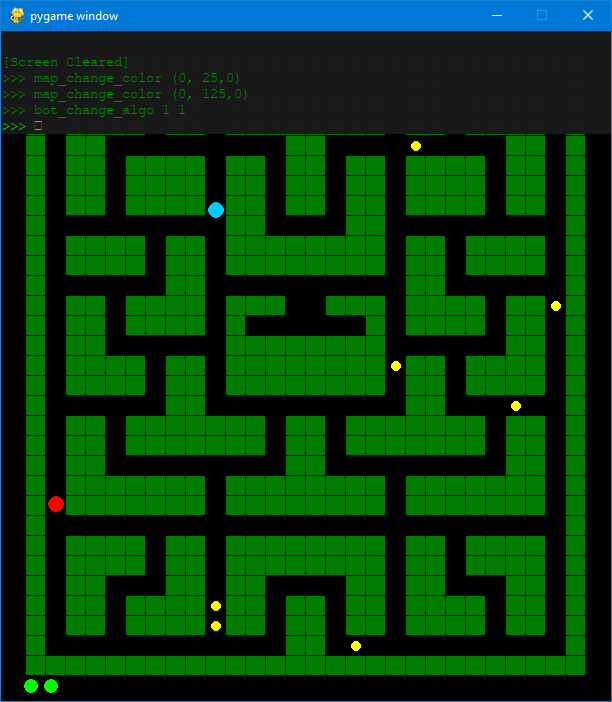
Вид ігрової консолі розробника

**Робота гри**

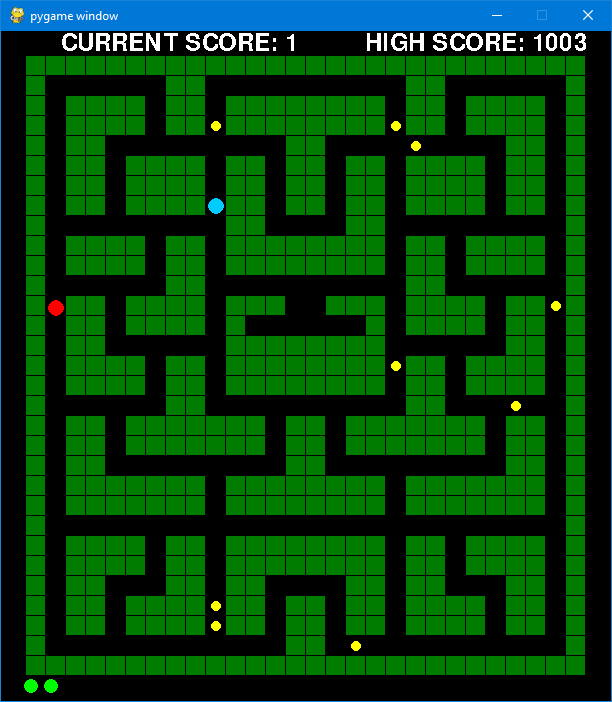
Можемо змінити колір карти за допомогою команди map\_change\_color (R, G,B)



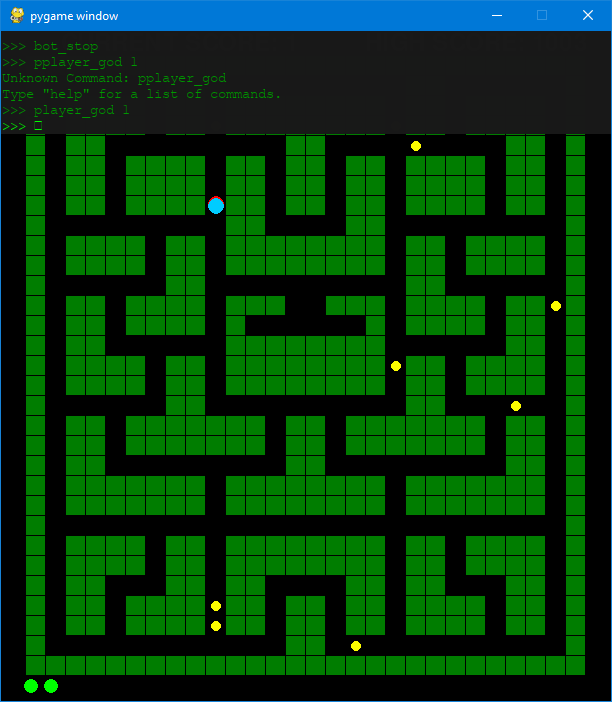
Існує функція для зміни поведінки бота за допомогою команди bot\_change\_algo BOT ALGO



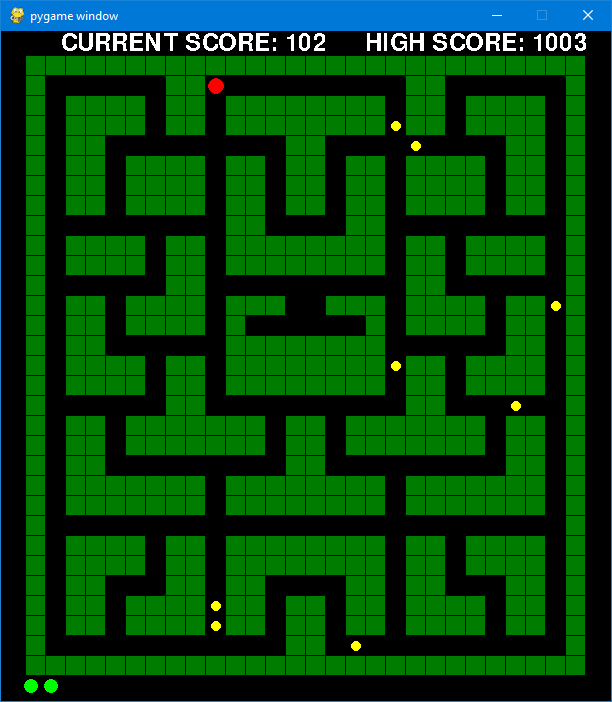
Можемо зупинити усіх ботів за допомогою команди bot\_stop



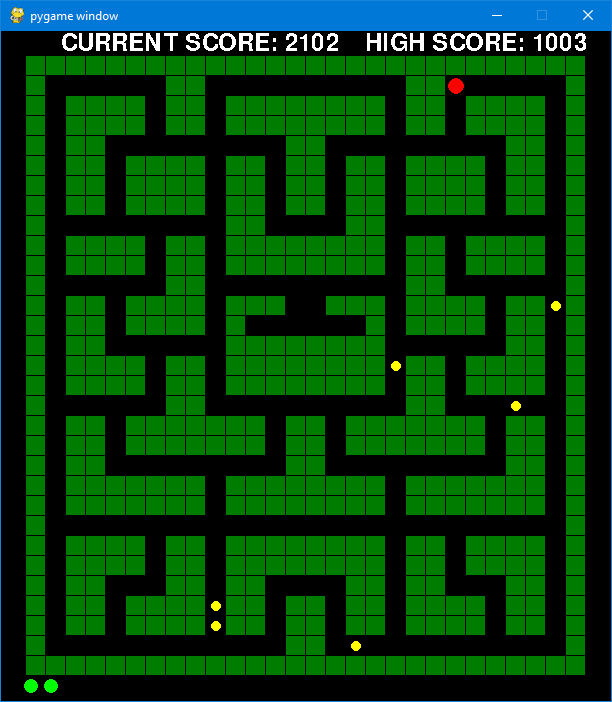
Можна зробити так, щоб пак-мен був у рожимі Бога та його не можна було з’їсти: player\_god 1



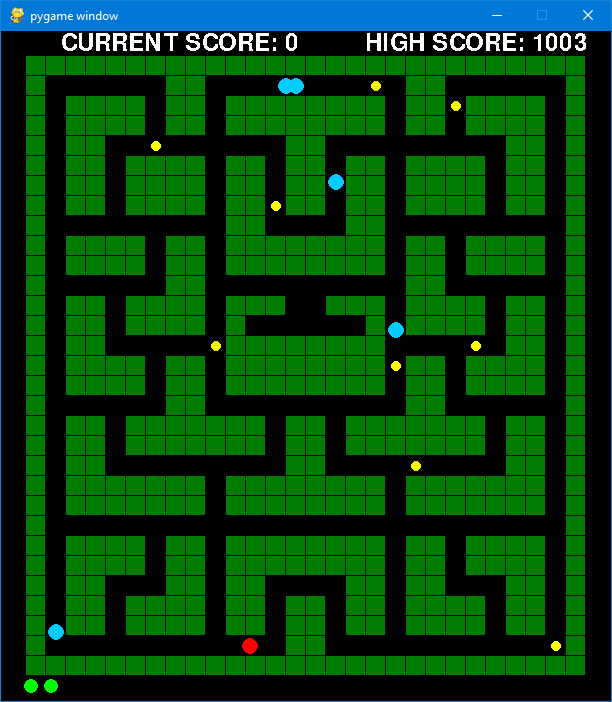
Можна зробити так, що б пакмен їв привидів: player\_can\_eat\_enemies



Можемо встановити ціну монетки: player\_set\_score\_for\_coin



Можемо додати ботів (не більше 4 за 1 раз): bot\_add NUMBER



**Висновок**

В даній роботі я, використовуючи консоль розробника, зробив декілька функцій для режиму god-mode в власній грі пак-мен.