Nr	Obszar	Wymaganie	KOD		Przyznane pkt	Pkt max
1	UI	JEST		☐		
		Wprowadzanie danych	def handle_events(self): for event in pygame.event.get(): if event.type == pygame.QUIT: pygame.quit() sys.exit() elif event.type == pygame.KEYDOWN: if event.key == pygame.K_ESCAPE: self.paused = not self.paused elif event.key == pygame.K_r and self.game_over: self.reset() elif not self.game_over and not self.paused: if event.key == pygame.K_UP and self.snake.direction != 'DOWN': self.snake.direction = 'UP' elif event.key == pygame.K_DOWN and self.snake.direction != 'UP': self.snake.direction = 'DOWN' elif event.key == pygame.K_LEFT and self.snake.direction != 'RIGHT': self.snake.direction = 'LEFT' elif event.key == pygame.K_RIGHT and self.snake.direction != 'LEFT': self.snake.direction = 'RIGHT'	✓		2
		Wyświetlanie danych	draw_text(self.screen, f"Score: {self.score}", self.colors['text'], 20, (10, 10)	✓		2

		Zmiana danych	def update(self): if self.game_over or self.paused: return
			
			`self.snake.move()`
			
			`# Sprawdzenie kolizji z jedzeniem`
			`if self.snake.body[0] == self.food.position:`
			`  self.snake.grow()`
			`  self.score += 10`
			`  self.food.spawn(self.snake.body)`
			
			`  # Zwiększenie prędkości co 50 punktów`
			`  if self.score % 50 == 0:`
			`  self.fps += 2`
				✓		2

			`# Sprawdzenie kolizji ze ścianami lub własnym ciałem`
			`if (self.snake.check_collision() or`
			`self.snake.body[0][0] < 0 or`
			`self.snake.body[0][0] >= self.grid_width or`
			`self.snake.body[0][1] < 0 or`
			`self.snake.body[0][1] >= self.grid_height):`
			`self.game_over = True`
			
			`self.high_scores.add_score(self.score)`			

		Wyszukiwanie danych	`def spawn(self, snake_body):`
			`available_positions = [`
			`(x, y) for x in range(self.grid_width)`
			`for y in range(self.grid_height)`
			`if (x, y) not in snake_body`
			`]`
				☐		2

		Przedstawienie wyników	draw_text(self.screen, f"Score: {self.score}", self.colors['text'], 20, (10, 10))	✓		2
2	Podstawy	Zmienne	class Game: def **init**(self, screen, block_size, colors, fps): self.screen = screen self.block_size = block_size self.colors = colors self.fps = fps self.clock = pygame.time.Clock() self.width, self.height = screen.get_size() self.grid_width = self.width // block_size self.grid_height = self.height // block_size self.snake = Snake(self.grid_width, self.grid_height, block_size) self.food = Food(self.grid_width, self.grid_height, block_size, self.snake.body) self.score = 0 self.game_over = False self.paused = False self.high_scores = HighScores('data/scores.json')	✓		2
		typy danych	def reset(self): start_x = self.grid_width // 2 start_y = self.grid_height // 2 self.body = [(start_x, start_y)] self.direction = random.choice(['UP', 'DOWN', 'LEFT', 'RIGHT']) self.grow_length = 0	✓		2

komentarze	""" Package główny gry Snake. Zawiera wszystkie moduły niezbędne do działania gry: - game.py - główna logika gry - snake.py - implementacja węża - food.py - implementacja jedzenia - high_scores.py - zarządzanie wynikami """	✓		1
operatory	self.score += 10 w Game.update()	✓		1,5
Instrukcje warunkowe (if, elif, else)	if self.game_over or self.paused:return	✓		3
Instrukcje iteracyjne				
for	def handle_events(self): for event in pygame.event.get(): if event.type == pygame.QUIT: pygame.quit() sys.exit() elif event.type == pygame.KEYDOWN: if event.key == pygame.K_ESCAPE: self.paused = not self.paused elif event.key == pygame.K_r and self.game_over: self.reset() elif not self.game_over and not self.paused: if event.key == pygame.K_UP and self.snake.direction != 'DOWN': self.snake.direction = 'UP' elif event.key == pygame.K_DOWN and self.snake.direction != 'UP': self.snake.direction = 'DOWN' elif event.key == pygame.K_LEFT and self.snake.direction != 'RIGHT': self.snake.direction = 'LEFT' elif event.key == pygame.K_RIGHT and self.snake.direction != 'LEFT': self.snake.direction = 'RIGHT'	✓		2
while	Game.run()	✓		2
Operacje wejścia (input)	def handle_events(self): for event in pygame.event.get(): if event.type == pygame.QUIT: pygame.quit() sys.exit() elif event.type == pygame.KEYDOWN: if event.key == pygame.K_ESCAPE: self.paused = not self.paused elif event.key == pygame.K_r and self.game_over: self.reset() elif not self.game_over and not self.paused: if event.key == pygame.K_UP and self.snake.direction != 'DOWN': self.snake.direction = 'UP' elif event.key == pygame.K_DOWN and self.snake.direction !=	✓		1,5

			'UP': self.snake.direction = 'DOWN' elif event.key == pygame.K_LEFT and self.snake.direction != 'RIGHT': self.snake.direction = 'LEFT' elif event.key == pygame.K_RIGHT and self.snake.direction != 'LEFT': self.snake.direction = 'RIGHT'			
		Operacje wyjścia (print)	draw_text(self.screen, f"Score: {self.score}", self.colors['text'], 20, (10, 10)	✓		1,5
		Funkcje z parametrami i wartościami zwracanymi	def get_highest_score(self): return max(self.scores) if self.scores else 0	✓		2
		Funkcje rekurencyjne	def get_average_score(self): if not self.scores: return 0 total = reduce(lambda x, y: x + y, self.scores) return total / len(self.scores)	✓		3
		Funkcje przyjmujące inne funkcje jako argumenty	def get_average_score(self): if not self.scores: return 0 total = reduce(lambda x, y: x + y, self.scores) return total / len(self.scores)	✓		3
		Dekoratory		☐		1,5
3	Kontenery	Użycie listy	class HighScores: def **init**(self, filename): self.filename = filename self.scores = self.load_scores()	✓		2
		Użycie słownika	class Game: def **init**(self, screen, block_size, colors, fps): self.screen = screen self.block_size = block_size self.colors = colors self.fps = fps self.clock = pygame.time.Clock() self.width, self.height = screen.get_size() self.grid_width = self.width // block_size self.grid_height = self.height // block_size self.snake = Snake(self.grid_width, self.grid_height, block_size) self.food = Food(self.grid_width, self.grid_height, block_size, self.snake.body) self.score = 0 self.game_over = False self.paused = False self.high_scores = HighScores('data/scores.json')	✓		2
		Użycie zbioru	nie używane bezpośrednio, ale podobna funkcjonalność w Food.spawn()	✓		1,5

		Użycie krotki	class Food: def **init**(self, grid_width, grid_height, block_size, snake_body): self.grid_width = grid_width self.grid_height = grid_height self.block_size = block_size self.position = (0, 0) # Initial position (will be updated by spawn) self.spawn(snake_body) # Spawn food in a random position at creation	✓		1,5
4	Przestrzenie nazw	Zastosowano zmienne lokalne	np. available_positions w Food.spawn()	✓		1,5
		Zastosowano zmienne globalne		☐		1,5
		Zastosowano zakresy funkcji		☐		1,5
		Zastosowano zakresy klas	class Food: def **init**(self, grid_width, grid_height, block_size, snake_body): self.grid_width = grid_width self.grid_height = grid_height self.block_size = block_size self.position = (0, 0) # Initial position (will be updated by spawn) self.spawn(snake_body) # Spawn food in a random position at creation	✓		1,5
5	Moduły i pakiety	Projekt podzielony na moduły (import, __init__)	from .game import Game from .snake import Snake from .food import Food from .high_scores import HighScores **all** = ['Game', 'Snake', 'Food', 'HighScores']	✓		2

Nr	Obszar	Wymaganie	KOD		Przyznane pkt	Pkt max

		Własne pakiety/funkcje pomocnicze w osobnych plikach .py	import matplotlib.pyplot as plt import os
			
			def plot_scores(scores): """Generuje i zapisuje wykres historii wyników gry.
			
			Args:
			    scores (list): Lista przechowująca historię wyników
			"""
			if not scores: # Jeśli brak wyników, zakończ funkcję
			    return
			
			# Konfiguracja wykresu
			plt.figure(figsize=(10, 5))
			plt.plot(scores, marker='o', linestyle='-', color='b')
			plt.title('Historia wyników gry', fontsize=14)
			plt.xlabel('Numer gry', fontsize=12)
			plt.ylabel('Wynik', fontsize=12)
			plt.grid(True, linestyle='--', alpha=0.7)
			
			# Utwórz katalog jeśli nie istnieje
				✓		2

			os.makedirs('assets', exist_ok=True)
			
			# Zapisz wykres
			plt.savefig('assets/wykres_wynikow.png', dpi=300, bbox_inches='tight')
			plt.close()			

6	Obsługa błędów	Obsługa wyjątków (try, except, finally)	def load_scores(self):
			
			try:
			
			with open(self.filename, 'r') as f:
			
			return json.load(f)
			
			except (FileNotFoundError, json.JSONDecodeError):
			
			return []
				✓		2

		Użycie assert do testów i walidacji	def test_initial_length(self): self.assertEqual(len(self.snake.body), 1)	✓		1,5
7	Łańcuchy znaków	Operacje na stringach (m.in. formatowanie, dzielenie, wyszukiwanie)	draw_text(self.screen, f"Score: {self.score}", self.colors['text'], 20, (10, 10))	✓		2

8	Obsługa plików	Odczyt z plików .txt, .csv, .json, .xml (min. 1)	def load_scores(self):
			
			try:
			
			with open(self.filename, 'r') as f:
			
			return json.load(f)
			
			except (FileNotFoundError, json.JSONDecodeError):
			
			return []
				✓		2

		Zapis do plików .txt, .csv, .json, .xml (min. 1)	def save_scores(self): with open(self.filename, 'w') as f: json.dump(self.scores, f, indent=4)	✓		2
9	OOP	Klasy	class Food: def **init**(self, grid_width, grid_height, block_size, snake_body): self.grid_width = grid_width self.grid_height = grid_height self.block_size = block_size self.position = (0, 0) self.spawn(snake_body) def spawn(self, snake_body): available_positions = [(x, y) for x in	✓		2

			range(self.grid_width) for y in range(self.grid_height) if (x, y) not in snake_body] if available_positions: self.position = random.choice(available_positions) else: self.position = (-1, -1) # No available positions			
		Metody	class Food: def **init**(self, grid_width, grid_height, block_size, snake_body): self.grid_width = grid_width self.grid_height = grid_height self.block_size = block_size self.position = (0, 0) self.spawn(snake_body) def spawn(self, snake_body): available_positions = [(x, y) for x in range(self.grid_width) for y in range(self.grid_height) if (x, y) not in snake_body] if available_positions: self.position = random.choice(available_positions) else: self.position = (-1, -1) # No available positions	✓		2
		Konstruktory	class Food: def **init**(self, grid_width, grid_height, block_size, snake_body): self.grid_width = grid_width self.grid_height = grid_height self.block_size = block_size self.position = (0, 0) self.spawn(snake_body)	✓		2
		Dziedziczenie		☐		2
10	Programowanie funkcyjne	map		☐		1,5
		filter		☐		1,5
		lambda	def get_average_score(self): if not self.scores: return 0 total = reduce(lambda x, y: x + y, self.scores) return total / len(self.scores)	✓		1,5
		reduce	def get_average_score(self): if not self.scores: return 0 total = reduce(lambda x, y: x + y, self.scores) return total / len(self.scores)	✓		1,5
11	Wizualizacja danych	Wygenerowano wykres (np. matplotlib, seaborn)	def plot_scores(scores): if not scores: return	✓		2

			plt.figure(figsize=(10, 5)) plt.plot(scores, marker='o') plt.title('High Scores History') plt.xlabel('Game') plt.ylabel('Score') plt.grid(True) if not os.path.exists('assets'): os.makedirs('assets') plt.savefig('assets/plot.png') plt.close()			

		Zapisano wykres do pliku graficznego (.png lub .jpg)	def plot_scores(scores):
			if not scores:
			return
			
			plt.figure(figsize=(10, 5))
			plt.plot(scores, marker='o')
			plt.title('High Scores History')
			plt.xlabel('Game')
			plt.ylabel('Score')
			plt.grid(True)
			
			if not os.path.exists('assets'):
			os.makedirs('assets')
			
			plt.savefig('assets/plot.png')
			plt.close()
				✓		1,5

T12	Testowanie	Testy jednostkowe (assert, unittest, pytest)	def test_initial_length(self): self.assertEqual(len(self.snake.body), 1)	✓		1,5
		Testy funkcjonalne	def test_initial_length(self): self.assertEqual(len(self.snake.body), 1)	✓		1,5
		Testy Integracyjne	def test_move_and_grow_integration(self): self.snake.grow() self.snake.grow() self.snake.move() self.snake.move() self.assertEqual(len(self.snake.body), 3)	✓		1,5
		Testy graniczne / błędne dane	def test_grow_to_maximum_size(self): for _ in range(400): # 20x20 grid (assuming one segment per block) self.snake.grow() self.snake.move() self.assertEqual(len(self.snake.body), 401) # 400 growths + initial	✓		1,5
		Testy wydajności (np. czas wykonania, timeit)	def test_performance_large_snake(self): for _ in range(10000): self.snake.grow() start = time.time() self.snake.move() end = time.time() self.assertLess(end - start, 0.1) # powinno wykonać się w < 100 ms	✓		1,5

		Testy pamięci memory_profiler	def test_memory_usage(self): tracemalloc.start() for _ in range(10000): self.snake.grow() current, peak = tracemalloc.get_traced_memory() tracemalloc.stop() self.assertLess(peak, 10**7) # mniej niż 10 MB pamięci	✓		1,5
		Test jakości kodu (flake8, pylint)	def test_no_duplicate_body_segments(self): self.snake.body = [(5, 5), (5, 6), (5, 7)] self.assertEqual(len(set(self.snake.body)), len(self.snake.body))	✓		1,5
13	Wersjonowanie	Repozytorium GIT		✓		1
		Historia commitów		✓		1
Nr	Obszar	Wymaganie	KOD		Przyznane pkt	Pkt max
		Link do GitHub	https://github.com/Psikutnik/Gra_typu_waz	✓		1
		Opis commitów		✓		1
14	Dokumentacja	Plik README.md (cel, autorzy, uruchamianie)		✓		1,5
		Przykładowe dane wejściowe i wyjściowe		☐		2
		Diagram klas lub struktura modułów		☐		2
			SUMA			