



## Zadanie 3.1

### 3. Wątki

#### 1. Opis ogólny

Treść Pliki Historia ? Pomoc

Termin 🕒 2021-02-24 zostało 60 dni	Trudność ☆☆☆☆☆ Punkty: 0	Próba -	Ocena maszynowa 🕒 ⚙️ ⚙️ ⚙️ Czas testu: 100sek	Inspekcja kodu ✓ ⚙️ ✖️ 🔒	Plagiat 🕒 ⚙️ ⚙️ ⚙️	Raporty Raport główny
--	--------------------------------	------------	---	-----------------------------	-----------------------	--------------------------

## Gra multiplayer z wykorzystaniem mechanizmów IPC

Poniżej opis dotyczy całej gry oraz jej mechaniki, wraz z opisem ogólnych wymagań z zakresu SO2.

Szczegóły dla ocen 3,0, 4,0 oraz 5,0 znajdują się odpowiednio w zadaniach 3.2 Specyfikacja na ocenę 3,0, 3.3 Specyfikacja na ocenę 4,0 oraz 3.4 Specyfikacja na ocenę 5,0.

Precyzują one niniejszy opis oraz są względem niego nadrzędne (np. mówią o braku konieczności implementacji niektórych elementów, danych w niniejszym opisie).

Istotne zmiany względem poprzednich projektów:

- Aby uzyskać ocenę 4,0 oraz 5,0 nie jest konieczne przysyłanie odpowiedzi na niższe oceny.
- Ze względu na obecność rozbudowanego interfejsu użytkownika, odpowiedzi w tym projekcie są oceniane **wyłącznie** przez Prowadzącego laboratorium. Przesłane odpowiedzi nie są poddawane testowaniu automatycznemu.
- Przed przystąpieniem do oceny należy **bez wyjątku** przesłać swoje rozwiązanie, na wybraną ocenę, do systemu Dante.
- Zaliczenie odbędzie się w terminie indywidualnie określonym przez Prowadzącego grupę.

## Wstęp

W ramach projektu z komunikacji międzyprocesowej należy przygotować prostą grę planszową dla czterech niezależnych graczy, grających równolegle. Architekturę gry należy zaprojektować tak, aby gracze-procesy komunikowały się z serwerem-procesem. Zarówno gracze, jak i serwer mają być oddzielnymi procesami.

Ponieważ graczem może być zarówno komputer jak i człowiek to sugerowane jest, aby całość składała się z trzech programów:

- serwera gry,
- klienta gracza,
- klienta bota.

## Specyfikacja gry

Gracze (🧑🏻, 🧑🏻, 🧑🏻, 🧑🏻) zamknęci są w labiryncie i mają za zadanie zbierać pojawiające się skarby w postaci monet (💰, 🪙, 🪙). Gracz, który zbierze wystarczająco dużo skarbów, zanoś je do obozowiska (🏠) i tam zostawia. Gracz może w danej chwili nosić dowolną liczbę monet (carried) ale może je stracić w wyniku ataku dzięki bestii (🐉) lub poprzez zderzenie się z innym graczem.

- W przypadku ataku dzięki bestii gracz ginie (death), a zebrany przez niego łup zostaje w miejscu śmierci (🏠). Gracz respawnuje się w swoim punkcie startowym.
- W przypadku zderzenia z innym graczem łupy obu pozostają w miejscu zderzenia (🏠), a gracze respawnują się w swoich punktach startowych.
- Pozostawiony łup (🏠) ma swoją wartość. W przypadku zderzenia jest to suma noszonych monet obu graczy.

Gracz pozbawia się swoich monet w obozowisku (🏠), gdzie zapisywane są one na jego konto (budget). Po zdaniu skarbu gracz kontynuuje poszukiwania, zaczynając od obozowiska.

## Typy graczy

Należy przygotować dwa typy graczy: komputer – bot (CPU) oraz człowiek (HUMAN).

- Postać **gracza-bota** porusza się autonomicznie na podstawie mapy przekazywanej przez serwer.
- Postać **gracza-człowieka** poruszana jest za pomocą klawiszy - strzałek (w górę, w dół, w lewo, w prawo).

Gracz CPU może poruszać się z wykorzystaniem dowolnego algorytmu, np. chaotycznie, A\* w dowolny punkt, A\* z eksploracją, RL, lewa ściana, itp. Gracz CPU powinien również reagować na pojawienie się bestii, serwując się ucieczką.

## Przeciwnicy

W grze należy przewidzieć co najmniej jeden typ przeciwnika – "dziką bestię" (🐉), która ma zachowywać się w następujący sposób:

- Jeżeli gracz nie ma w polu widzenia bestii, to może ona beczelowo poruszać się w dowolnym kierunku, lub stać w miejscu (lub łączyć te dwa zachowania).
- Jeżeli gracz pojawił się w polu widzenia, to bestia podejmuje pościg, aż do ucieczki gracza z pola widzenia. Po zgubieniu gracza bestia może wrócić do beczelowego poruszania się lub dotrzeć do miejsca, w którym ostatnio widziała gracza.
  - Zgubienie gracza ma miejsce tylko wtedy, gdy linia prosta między środkiem postaci gracza a środkiem postaci bestii jest przecięta przez obiekt ściany. Przykład: A) bestia widzi gracza (pościgi); B) bestia nie widzi gracza; C) bestia widzi gracza; D) bestia widzi gracza, ale nie ma jak podjąć pościgu.



## Pole widzenia

Serwer nie przekazuje graczom kompletnej mapy w jakikolwiek sposób. Przekazuje jedynie dane w ramach pola widzenia gracza. Przez pole widzenia gracza należy rozumieć obszar o promieniu dwóch pól dookoła aktualnej pozycji postaci gracza (patrz Widok gracza). Wraz z polem widzenia, gracz otrzymuje swoją pozycję w świecie (taki GPS). Graczowi nie wolno zakładać jakichkolwiek informacji o świecie. Jedyne dopuszczalne założenie to maksymalny rozmiar świata – 128x128.

Wraz z informacjami o polu widzenia, gracz otrzymuje od serwera swoje współrzędne. Informacje te można wykorzystać do budowy mapy świata po stronie gracza.

## Dołączanie do i porzucanie gry

Serwer może obsługiwać maksymalnie czterech graczy. Jeżeli użytkownik spróbuje uruchomić kolejnego gracza (proces gracza/bota), to ten ma wyświetlić komunikat o pełnym serwerze i zakończyć swoje działanie. Dla każdego, nowo przybyłego gracza, losowane jest miejsce (współrzędne x,y) spawnu. W tym miejscu postać gracza pojawia się przy starcie gry oraz po każdej śmierci.

Gracz-człowiek, jak i gracz-bot, mają być traktowani tak samo. Z punktu widzenia serwera nie może istnieć faworyzowanie jednego bądź drugiego. **Ma to wymuszać wspólny interfejs API między serwerem a jego klientami.**

Serwer powinien umieć radzić sobie z sytuacją, gdy gracz opuszcza grę w naturalny sposób (np. wciskając q po stronie klienta) lub nagły sposób (zabicie procesu klienta).

## Podział czasu gry

Gra składa się z tur, z których każda ma trwać nie więcej niż sekundę (sugerowana parametryzacja na czas pisania kodu). W ramach jednej tury każda postać (bestia/gracz/bot) może przemieścić się nie więcej niż o jedno pole. Może też stać w miejscu. Ruch ma odbywać się na zasadzie informowania serwera o chęci zrobienia kroku w jednym z czterech kierunków. Serwer ma prawo odmówić wykonania ruchu (np. jeżeli gracz będzie chciał przejść przez ścianę).

Jeżeli klient nie zadeklaruje chęci ruchu w danej turze, to gra jest kontynuowana. Ułatwi to proces debugowania klientów, gdy jeden z nich jest wstrzymany w trybie debugowania.

## Przeszkody

W grze należy zaimplementować co najmniej dwa typy przeszkód: ścianę (■) oraz krzaki (s). Postać nie może przejść przez ścianę – serwer nie może od tego dopuścić. Przejście przez krzaki jest możliwe, ale wymaga to **dwoch** tur, w ramach których gracz próbuje przejść przez przeszkodę.

### Statystyki

Wszelkie statystyki przechowywane i liczone są po stronie serwera. Gracz jest jedynie informowany o ich wartościach. Dzięki temu gracz nie może ich "sztucznie" zmienić (np. dodając pewną liczbę monet).

### Widok serwera

Parameter:	Player1	Player2	Player3	Player4
PID	1390	1420	1900	-
Type	CPU	CPU	HUMAN	-
Curr X/Y	37/15	16/21	04/03	--/--
Deaths	3	9	0	-

Coins	Player1	Player2	Player3	Player4
carried	30	213	0	-
brought	1300	33	0	-

Legend:

- - players
- - wall
- \* - bushes (slow down)
- - wild beast
- c - one coin
- t - treasure (10 coins)
- T - large treasure (50 coins)
- A - campsite
- - dropped treasure

Serwer musi, równoległe z grą obsługiwać klawiaturę, dopuszczając następujące reakcje na wciskane klawisze:

- **b/b** – dodanie jednej bestii w losowym miejscu labiryntu,
- **c/t/t** – dodanie nowej monety, skarbu, dużego skarbu w losowym miejscu labiryntu,
- **q/q** – zakończenie gry.

### Widok gracza:

Player:

Number: 3

Type: HUMAN

Curr X/Y: 04/03

Deaths: 0

Coins found: 0

Coins brought: 0

Legend:

- - players
- - wall
- \* - bushes (slow down)
- - enemy
- c - one coin
- t - treasure (10 coins)
- T - large treasure (50 coins)
- A - campsite
- - dropped treasure

Klient gracza-człowieka powinien obsługiwać klawisze strzałek (góra, dół, lewo, prawo) do poruszania postacią gracza. Ponadto klienty gracza-człowieka i bota powinny dawać możliwość kontrolowanego zakończenia gry (klawisz **q**). **Zabicie procesu gracza nie jest kontrolowanym sposobem zakończenia gry.**

### Warunki zaliczenia

- Serwer oraz klienty muszą komunikować się ze sobą jedynie za pomocą mechanizmów IPC w ramach tej samej maszyny. Niedopuszczalne jest wymienianie danych poprzez fizyczne pliki.
- Niedopuszczalne jest aktywne oczekiwanie, marnujące czas procesora; należy korzystać z funkcji blokujących, np. `sem_wait()` albo `sem_trywait()`. Niedopuszczalne są realizacje oczekiwania poprzez `spinlock`.
  - Dla przykładu próbkowanie semafora funkcją `sem_getvalue` i podejmowanie decyzji na podstawie odczytanej wartości **jest** spinlockiem.
- Kolejność uruchamiania procesów składowych nie może mieć znaczenia.
- Nie wolno stosować gotowych bibliotek, opakowujących systemowe mechanizmy IPC. Jeżeli uważasz, że znaleziona przez Ciebie biblioteka jest w porządku i chcesz ją wykorzystać – uzgodnij z koordynatorem przedmiotu SO2. Przykład: Wykorzystywanie metod typu `readInt` z klas strumieni, dziedziczących po `DataStream` **jest** niedopuszczalne. Mechanizmy te implementują, w sposób niewidoczny dla programisty, obsługę sytuacji w których odbiornik nie otrzymał jeszcze kompletu danych (np. dostał dopiero 3 z 4 bajtów inta).
- Klienty nie mogą operować we wspólnej przestrzeni pamięci – klient nie może mieć dostępu (nawet, jeżeli z niego nie korzysta) do danych innych klientów.
- Serwer oraz klienty nie mogą pozostawiać po sobie żadnych aktywnych/otwartych semaforów oraz plików.

#### Podpowiedzi

- Czy gracz-bot różni się czymś szczególnym od bota bestii?
- Programy nie muszą być napisane w tych samych językach.