Lista 2

Termin wykonania: 2025-05-04

Wymagania co do zgłaszanych rozwiązań:

- Implementacja nie może wprowadzać sztucznych zależności. Zdarzenia niezależne w symulowanym świecie (jak na przykład przeprowadzki w odległych fragmentach kraty) powinny się wykonywać współbieżnie w symulacji, bez sytuacji, że akcje jednego z nich mogą wstrzymywać jakieś akcje drugiego.
- Prezentacje przygotować zgodnie z wymaganiami dla dla zadań z Listy 1. Jeśli przyjmujemy, że jakiś obiekt znika z planszy (zwalnia pole), to powinien wtedy zostawić ślad ze współrzędnymi gdzieś poza planszą (np. współrzędne `(Board_Width, Board_Height)`), aby zwolnione pole pojawiło się jako '`.`'.
- Jakość prezentacji będzie mieć wpływ na ocenę.

Zadanie 1. (za 10 punktów)

Przerobić system zaimplementowany w Adzie jako rozwiązanie Zadania 2 z Listy 1, rozszerzając go w następujący sposób:

- W każdym wolnym polu kraty, w losowym momencie, może pojawić się *dziki lokator* o następujących własnościach:
  - Ma ograniczony czas życia, po którym znika.
  - Zajmuje miejsce na polu.
  - Symbolem dzikiego lokatora jest cyfra dziesiętna.
- Jeśli legalny podróżnik p chce przejść z pola v na pole, zajmowane przez dzikiego lokatora, to dziki lokator musi się wyprowadzić do sąsiedniego pustego pola różnego od v, o ile takie pole istnieje. Jeśli nie istnieje, to p nie może się przeprowadzić i w następnym kroku ponownie losuje kierunek ruchu.
- Na potrzeby skryptu `display-travel.bash`, dziki lokator, podobnie jak podróżnik, notuje swoje ślady przy zmianach stanów i na końcu przekazuje je do wątku `Printer`.
- Każde pole obsługiwane jest przez osobny wątek, tak aby były spełnione powyższe założenia. Podróżnik mógłby na przykład zgłaszać prośbę o przeprowadzkę do wątku obsługującego jego bieżące pole, który może kontaktować się z wątkami innych pól aby dokonać takiej przeprowadzki z sposób "legalny" albo uzyskać informację dla podróżnika, że przeprowadzka nie jest możliwa.

Zadanie 2. (za 10 punktów)

Zaimplementować system z Zadania 1 w języku Go. Do implementacji wątków obsługujących pola zastosować uogólnienie mechanizmu wzorowanego na Stateful Goroutines. (Pomyśl o wątku obsługującym pole, jako o serwerze obsługującym prośby podróżników i sąsiednich pól, który sam może stać się klientem sąsiednich pól aby jakąś prośbę zrealizować.)

Zadanie 3. (za 10 punktów)

Rozszerz system z Zadania 1 w następujący sposób:

- Pewna liczba losowo wybranych pól stanowi *pułapki*.
- Dzicy lokatorzy nie rozpoczynają swego istnienia w polach-pułapkach.
- Pułapka ma symbol: '#'.
- Podobnie jak podróżnik, pułapka (na potrzeby skryptu `display-travel.bash`) notuje swoje ślady:
    - pierwszy - zanim wątki podróżników zostaną uruchomione, a następne
    - w istotnych zdarzeniach opisanych niżej.
- Jeśli podróżnik p wejdzie na pole-pułapkę, to:
    - zmienia swój symbol na małą literę, odnotowuje swój ślad i na chwilę zasypia blokując to pole,
    - następnie wątek podróżnika się kończy, zwalnia pole a pułapka odnotowuje swój ślad aby w tym momencie ponownie pojawiła się jako '#'.
- Jeśli dziki lokator wejdzie na pole-pułapkę, to:
    - zmienia swój symbol na '*', odnotowuje swój ślad i na chwilę zasypia blokując to pole,
    - następnie zwalnia pole, kończy życie a pułapka odnotowuje swój ślad z symbolem '#'.
- Zauważ, że pola-pułapki muszą przekazać ślady swoich zdarzeń do wątku `Printer` po zakończeniu wątków podróżników i dzikich lokatorów.

Zadanie 4. (za 10 punktów)

Zaimplementuj system z Zadania 3 w języku Go.