K - Konstruktorzy
C - Nierozróżnialne części

Każdy konstruktor:

1. Tworzenie projektu (losowy czas)
2. Pobranie losowej liczby części C
3. Budowanie robota (losowy czas)
4. Dobranie się w parę z innym konstruktorem K
5. Walka (losowy czas)
6. Zwrócenie części części C
7. Zwrócenie reszty części C

1. Struktury i zmienne procesu i:
    a. C - liczba nierozróżnialnych części
    b. wanted - liczba części żądanych przez konstruktora, początkowo 0
    c. owned - liczba posiadanych części, początkowo 0
    d. taken - suma części zajętych przez innych konstruktorów, początkowo 0
    e. AckCounterTake - liczba otrzymanych potwierdzeń ACK_TAKE, początkowo 0
    f. ReceivedAckTake - wektor wartości true/false odpowiadających konstruktorom, w którym odznaczani są konstruktorzy, od których otrzymaliśmy ACK_TAKE po wysłaniu REQ_TAKE.
    g. FightQueue - lista oczekujących do walki posortowana względem priorytetów, początkowo pusta
    h. AckCounterFight - liczba otrzymanych potwierdzeń ACK_FIGHT, początkowo 0
    i. AckCounterOpponent - liczba otrzymanych potwierdzeń ACK_OPPONENT_FOUND, początkowo 0
    j. AckCounterReturn - liczba otrzymanych potwierdzeń ACK_RETURN, początkowo 0
    k. n - liczba konstruktorów
    l. FightBuffer - wektor konstruktorów od których nie otrzymaliśmy REQ_FIGHT, a którzy znaleźli się w wiadomości REQ_OPPONENT_FOUND

2. Wiadomości. Wszystkie wiadomości są podbite znacznikiem czasowym (timestampem), modyfikowanym zgodnie z zasadami skalarnego zegara logicznego Lamporta. :
    a. REQ_TAKE - żądanie o dostęp do sekcji krytycznej związanej z pobraniem części. Zawiera liczbę żądanych części
    b. ACK_TAKE - potwierdzenie otrzymania żądania części. Zawiera liczbę aktualnie posiadanych części
    c. REQ_FIGHT - poinformowanie o chęci do walki
    d. ACK_FIGHT - potwierdzenie wpisania na listę oczekujących do walki
    e. REQ_OPPONENT_FOUND - poinformowanie o wybraniu przeciwnika do walki. Zawiera identyfikator przeciwnika
    f. ACK_OPPONENT_FOUND - potwierdzenie otrzymania wiadomości o wybraniu przeciwnika przez konstruktora
    g. REQ_RETURN - informowanie o chęci zwrócenia części. Zawiera liczbę zwracanych części
    h. ACK_RETURN - potwierdzenie otrzymania żądania zwrócenia części

3. Stany:
    a. REST_PROJECT - początkowy stan procesu. Konstruktor jest w fazie projektowania robota, nie ubiega się o dostęp do sekcji krytycznej
    b. WAIT_TAKE - czeka na dostęp do sekcji krytycznej związanej z pobraniem części

c. INSECTION_TAKE - w sekcji krytycznej związanej z pobraniem części
d. REST_BUILDING - konstruktor buduje robota i przygotowuje się do walki
e. WAIT_FIGHT - konstruktor informuje o chęci wzięcia udziału w walce, czeka na potwierdzenie
f. INSECTION_FIGHT - konstruktor przebywa w sekcji krytycznej, walczy z przeciwnikiem
g. WAIT_RETURN - czeka na dostęp do sekcji krytycznej związanej ze zwróceniem części części
h. REST_REPAIR - konstruktor naprawia pozostałe części
i. WAIT_RETURN_REMAINING - czeka na dostęp do sekcji krytycznej związanej ze zwróceniem reszty części

4. Szkic algorytmu - Konstruktor i ubiegający się o wejście do sekcji krytycznej związanej z pobraniem części wysyła do wszystkich pozostałych prośby REQ_TAKE o dostęp. Pozostali konstruktorzy odsyłają ACK_TAKE do konstruktora. Konstruktor wchodzi do sekcji po zebraniu ACK_TAKE od wszystkich i upewnieniu się, że znajduje się dla niego wystarczająca liczba części do zabrania.

   Po budowie robota z części, konstruktorzy dobierają się w pary i walczą. Gdy konstruktor przechodzi w stan WAIT_FIGHT, wysyła wszystkim pozostałym konstruktorom żądanie REQ_FIGHT, aby zasygnalizować chęć do walki. Konstruktor tworzy sobie kolejkę chętnych do walki posortowaną według priorytetów żądań. Jeżeli jest na parzystym miejscu (licząc od 1), wysyła wszystkim informację o tym, że walczy z konstruktorem nad sobą oraz wchodzi do INSECTION_FIGHT.

   Aby oddać części, konstruktorzy wysyłają żądanie REQ_RETURN do wszystkich procesów, aby poinformować ich o zwróceniu części do bazy. Czekają na otrzymanie potwierdzeń ACK_RETURN, aby kontynuować swoje zadania. Po chwili powtarzają podobne żądania chcąc zwrócić naprawione części. Używają wtedy znowu REQ_RETURN i ACK_RETURN.

   Uwaga: Im większy zegar Lamporta, tym mniejszy priorytet.

5. Opis szczegółowy algorytmu dla procesu i:
   a. REST_PROJECT: stan początkowy. Konstruktor i przebywa w stanie REST_PROJECT do czasu, aż podejmie decyzję o ubieganie się o sekcję krytyczną, czyli o pobranie części potrzebnych do budowy robota. Konstruktor decyduje o liczbie potrzebnych części i zapisuje ją w zmiennej wanted. Aktualizuje wektor ReceivedAckTake[0..n] = false. Ze stanu REST_PROJECT następuje przejście do stanu WAIT_TAKE po uprzednim wysłaniu wiadomości REQ_TAKE do wszystkich innych konstruktorów oraz ustawieniu zmiennych AckCounterTake i taken na zero. Wszystkie wiadomości REQ_TAKE są opisane tym samym priorytetem, równym zegarowi Lamporta w chwili wysłania pierwszej wiadomości REQ_TAKE. Zawierają one również liczbę części żądanych przez konstruktora. Reakcje na wiadomości:
      i. REQ_TAKE: odpowiada ACK_TAKE z liczbą posiadanych przez siebie części, czyli z 0
      ii. ACK_TAKE: ignorowane
      iii. REQ_FIGHT: Jeżeli wartość w FightBuffer odpowiadająca konstruktorowi j jest równa 0 - odsyła ACK_FIGHT oraz dopisuje konstruktora do listy FightQueue w odpowiednie miejsce zależne od jego priorytetu. Jeżeli jest większa od 0, konstruktor i odejmuje 1 od tej wartości.
      iv. ACK_FIGHT: ignorowane

v. REQ_OPPONENT_FOUND: usuwa z kolejki zarówno konstruktora nadawcę jak i konstruktora o identyfikatorze zawartym w żądaniu. Jeżeli konstruktora o identyfikatorze zawartym w żądaniu nie było w kolejce, dodaje wartość 1 do pola odpowiadającego temu konstruktorowi w wektorze FightBuffer.
vi. ACK_OPPONENT_FOUND: ignorowane
vii. REQ_RETURN: odsyła ACK_RETURN
viii. ACK_RETURN: ignorowane

b. WAIT_TAKE: ubieganie się o sekcję krytyczną. Ze stanu WAIT_TAKE następuje przejście do stanu INSECTION_TAKE pod warunkiem, że konstruktor otrzyma ACK_TAKE od wszystkich innych konstruktorów (AckCounterTake == n - 1) oraz liczba dostępnych części jest przynajmniej tak duża jak liczba żądanych części, czyli C - taken >= wanted. Reakcje na wiadomości:
   i. REQ_TAKE: jeśli REQ_TAKE konstruktora i ma priorytet wyższy od otrzymanego REQ_TAKE od konstruktora j, to odsyłane jest ACK_TAKE z wartością wanted. W przeciwnym wypadku konstruktor i wysyła ACK_TAKE z wartością 0.
   ii. ACK_TAKE: zwiększa licznik otrzymanych ACK_TAKE (AckCounterTake++) oraz aktualizuje ReceivedAckTake[j] = true. Aktualizuje taken na podstawie części otrzymanych w wiadomości. Tak, jak opisano to wyżej, gdy AckCounterTake == n - 1 oraz C - taken >= wanted, konstruktor i przechodzi do stanu INSECTION_TAKE.
   iii. REQ_FIGHT: Jeżeli wartość w FightBuffer odpowiadająca konstruktorowi j jest równa 0 - odsyła ACK_FIGHT oraz dopisuje konstruktora do listy FightQueue w odpowiednie miejsce zależne od jego priorytetu. Jeżeli jest większa od 0, konstruktor i odejmuje 1 od tej wartości.
   iv. ACK_FIGHT: ignorowane
   v. REQ_OPPONENT_FOUND: usuwa z kolejki zarówno konstruktora nadawcę jak i konstruktora o identyfikatorze zawartym w żądaniu. Jeżeli konstruktora o identyfikatorze zawartym w żądaniu nie było w kolejce, dodaje wartość 1 do pola odpowiadającego temu konstruktorowi w wektorze FightBuffer.
   vi. ACK_OPPONENT_FOUND: ignorowane
   vii. REQ_RETURN: odsyła ACK_RETURN oraz jeżeli ReceivedAckTake[j] = true odejmuje liczbę części przekazaną w żądaniu od zmiennej taken. Jeżeli warunek C - taken >= wanted jest spełniony, wchodzi do INSECTION_TAKE
   viii. ACK_RETURN: ignorowane

c. INSECTION_TAKE: przebywanie w sekcji krytycznej związanej z pobraniem/zwróceniem części. Następuje zaktualizowanie zmiennej owned oznaczającej liczbę posiadanych przez siebie części. Konstruktor przebywa w sekcji krytycznej do czasu podjęcia decyzji o jej opuszczeniu. Po podjęciu decyzji o opuszczeniu sekcji, konstruktor przechodzi do stanu REST_BUILDING. Reakcje na wiadomości:
   i. REQ_TAKE: odpowiada ACK_TAKE z liczbą pobieranych przez siebie części
   ii. ACK_TAKE: ignorowane
   iii. REQ_FIGHT: Jeżeli wartość w FightBuffer odpowiadająca konstruktorowi j jest równa 0 - odsyła ACK_FIGHT oraz dopisuje konstruktora do listy FightQueue w odpowiednie miejsce zależne od jego priorytetu. Jeżeli jest większa od 0, konstruktor i odejmuje 1 od tej wartości.
   iv. ACK_FIGHT: ignorowane
   v. REQ_OPPONENT_FOUND: usuwa z kolejki zarówno konstruktora nadawcę jak i konstruktora o identyfikatorze zawartym w żądaniu. Jeżeli konstruktora o

identyfikatorze zawartym w żądaniu nie było w kolejce, dodaje wartość 1 do pola odpowiadającego temu konstruktorowi w wektorze FightBuffer.

vi. ACK_OPPONENT_FOUND: ignorowane
vii. REQ_RETURN: odsyła ACK_RETURN
viii. ACK_RETURN: ignorowane

d. REST_BUILDING: Konstruktor i przebywa w stanie REST_BUILDING do czasu, aż podejmie decyzję o ubieganie się o sekcję krytyczną, czyli o znalezienie przeciwnika i rozpoczęcie walki. Ze stanu REST_BUILDING następuje przejście do stanu WAIT_FIGHT po uprzednim wysłaniu wiadomości REQ_FIGHT do wszystkich innych konstruktorów oraz ustawieniu AckCounterFight na zero. Wszystkie wiadomości REQ_FIGHT są opisane tym samym priorytetem, równym zegarowi Lamporta w chwili wysłania pierwszej wiadomości REQ_FIGHT. Reakcje na wiadomości:
   i. REQ_TAKE: odpowiada ACK_TAKE z liczbą pobieranych przez siebie części
   ii. ACK_TAKE: ignorowane
   iii. REQ_FIGHT: Jeżeli wartość w FightBuffer odpowiadająca konstruktorowi j jest równa 0 - odsyła ACK_FIGHT oraz dopisuje konstruktora do listy FightQueue w odpowiednie miejsce zależne od jego priorytetu. Jeżeli jest większa od 0, konstruktor i odejmuje 1 od tej wartości.
   iv. ACK_FIGHT: ignorowane
   v. REQ_OPPONENT_FOUND: usuwa z kolejki zarówno konstruktora nadawcę jak i konstruktora o identyfikatorze zawartym w żądaniu. Jeżeli konstruktora o identyfikatorze zawartym w żądaniu nie było w kolejce, dodaje wartość 1 do pola odpowiadającego temu konstruktorowi w wektorze FightBuffer.
   vi. ACK_OPPONENT_FOUND: ignorowane
   vii. REQ_RETURN: odsyła ACK_RETURN
   viii. ACK_RETURN: ignorowane

e. WAIT_FIGHT: ubieganie się o sekcję krytyczną. Ze stanu WAIT_FIGHT następuje przejście do stanu INSECION_FIGHT pod warunkiem, że konstruktor otrzyma ACK_FIGHT od wszystkich innych konstruktorów (AckCounterFight == n - 1) oraz nastąpi jedna z dwóch sytuacji: Konstruktor wyśle żądanie REQ_OPPONENT_FOUND, a później otrzyma n - 1 ACK_OPPONENT_FOUND lub otrzyma REQ_OPPONENT_FOUND ze swoim identyfikatorem w treści. Reakcje na wiadomości:
   i. REQ_TAKE: odpowiada ACK_TAKE z liczbą posiadanych przez siebie części
   ii. ACK_TAKE: ignorowane
   iii. REQ_FIGHT: Jeżeli wartość w FightBuffer odpowiadająca konstruktorowi j jest równa 0 - odsyła ACK_FIGHT oraz dopisuje konstruktora do listy FightQueue w odpowiednie miejsce zależne od jego priorytetu. Jeżeli jest większa od 0, konstruktor i odejmuje 1 od tej wartości.
   iv. ACK_FIGHT: zwiększa zmienną AckCounterFight. Gdy wynosi ona n - 1 i konstruktor znajduje się na parzystej (w systemie liczenia od 1) pozycji, wysyła REQ_OPPONENT_FOUND z identyfikatorem konstruktora umiejscowionego bezpośrednio nad sobą w kolejce, ustawia AckCounterOpponent na 0
   v. REQ_OPPONENT_FOUND - odpowiada ACK_OPPONENT_FOUND. Jeżeli w żądaniu znajduje się identyfikator konstruktora i, przechodzi on do stanu INSECTION_FIGHT. Będzie odbywał walkę z konstruktorem, który jest autorem żądania. Jeżeli w żądaniu znajduje się identyfikator innego konstruktora, konstruktor i usuwa z kolejki zarówno konstruktora nadawcę jak i konstruktora o identyfikatorze zawartym w żądaniu. Jeżeli konstruktora o

identyfikatorze zawartym w żądaniu nie było w kolejce, dodaje wartość 1 do pola odpowiadającego temu konstruktorowi w wektorze FightBuffer.

vi. ACK_OPPONENT_FOUND - zwiększa zmienną AckCounterOpponent. Gdy wynosi ona n - 1, konstruktor przechodzi do kolejnego stanu
vii. REQ_RETURN: odsyła ACK_RETURN
viii. ACK_RETURN: ignorowane

f. INSECTION_FIGHT: przebywanie w sekcji krytycznej związanej z walką. Konstruktor przebywa w sekcji krytycznej do czasu zakończenia walki. Następnie przechodzi do stanu WAIT_RETURN po uprzednim rozesłaniu REQ_RETURN. Reakcje na wiadomości:
   i. REQ_TAKE: odpowiada ACK_TAKE z liczbą posiadanych przez siebie części
   ii. ACK_TAKE: ignorowane
   iii. REQ_FIGHT: Jeżeli wartość w FightBuffer odpowiadająca konstruktorowi j jest równa 0 - odsyła ACK_FIGHT oraz dopisuje konstruktora do listy FightQueue w odpowiednie miejsce zależne od jego priorytetu. Jeżeli jest większa od 0, konstruktor i odejmuje 1 od tej wartości.
   iv. ACK_FIGHT: ignorowane
   v. REQ_OPPONENT_FOUND - usuwa z kolejki zarówno konstruktora nadawcę jak i konstruktora o identyfikatorze zawartym w żądaniu. Jeżeli konstruktora o identyfikatorze zawartym w żądaniu nie było w kolejce, dodaje wartość 1 do pola odpowiadającego temu konstruktorowi w wektorze FightBuffer.
   vi. ACK_OPPONENT_FOUND - ignorowane
   vii. REQ_RETURN: odsyła ACK_RETURN
   viii. ACK_RETURN: ignorowane

g. WAIT_RETURN: informowanie pozostałych konstruktorów o oddaniu części. Ze stanu WAIT_RETURN następuje przejście do stanu REST_REPAIR pod warunkiem, że konstruktor otrzyma ACK_RETURN od wszystkich innych konstruktorów (AckCounterReturn == n - 1). Reakcje na wiadomości:
   i. REQ_TAKE: odpowiada ACK_TAKE z liczbą posiadanych przez siebie części
   ii. ACK_TAKE: ignorowane
   iii. REQ_FIGHT: Jeżeli wartość w FightBuffer odpowiadająca konstruktorowi j jest równa 0 - odsyła ACK_FIGHT oraz dopisuje konstruktora do listy FightQueue w odpowiednie miejsce zależne od jego priorytetu. Jeżeli jest większa od 0, konstruktor i odejmuje 1 od tej wartości.
   iv. ACK_FIGHT: ignorowane
   v. REQ_OPPONENT_FOUND - usuwa z kolejki zarówno konstruktora nadawcę jak i konstruktora o identyfikatorze zawartym w żądaniu. Jeżeli konstruktora o identyfikatorze zawartym w żądaniu nie było w kolejce, dodaje wartość 1 do pola odpowiadającego temu konstruktorowi w wektorze FightBuffer.
   vi. ACK_OPPONENT_FOUND - ignoruje
   vii. REQ_RETURN: odsyła ACK_RETURN
   viii. ACK_RETURN: zwiększa zmienną AckCounterReturn. Gdy jest ona równa liczbie n - 1, części zostają zwrócone

h. REST_REPAIR: Konstruktor i przebywa w stanie REST_REPAIR do czasu, aż podejmie decyzję o zwróceniu naprawionych części. Ze stanu REST_REPAIR następuje przejście do stanu WAIT_RETURN_REMAINING po uprzednim wysłaniu wiadomości REQ_RETURN do wszystkich innych konstruktorów oraz ustawieniu AckCounterReturn na zero. Reakcje na wiadomości:
   i. REQ_TAKE: odpowiada ACK_TAKE z liczbą posiadanych przez siebie części

ii. ACK_TAKE: ignorowane
iii. REQ_FIGHT: Jeżeli wartość w FightBuffer odpowiadająca konstruktorowi j jest równa 0 - odsyła ACK_FIGHT oraz dopisuje konstruktora do listy FightQueue w odpowiednie miejsce zależne od jego priorytetu. Jeżeli jest większa od 0, konstruktor i odejmuje 1 od tej wartości.
iv. ACK_FIGHT: ignorowane
v. REQ_OPPONENT_FOUND - usuwa z kolejki zarówno konstruktora nadawcę jak i konstruktora o identyfikatorze zawartym w żądaniu. Jeżeli konstruktora o identyfikatorze zawartym w żądaniu nie było w kolejce, dodaje wartość 1 do pola odpowiadającego temu konstruktorowi w wektorze FightBuffer.
vi. ACK_OPPONENT_FOUND - ignoruje
vii. REQ_RETURN: odsyła ACK_RETURN
viii. ACK_RETURN: ignorowane

i. WAIT_RETURN_REMAINING: informowanie pozostałych konstruktorów o oddaniu pozostałych części. Ze stanu WAIT_RETURN_REMAINING następuje przejście do stanu REST_PROJECT pod warunkiem, że konstruktor otrzyma ACK_RETURN od wszystkich innych konstruktorów (AckCounterReturn == n - 1). Reakcje na wiadomości:
    i. REQ_TAKE: odpowiada ACK_TAKE z liczbą posiadanych przez siebie części
    ii. ACK_TAKE: ignorowane
    iii. REQ_FIGHT: Jeżeli wartość w FightBuffer odpowiadająca konstruktorowi j jest równa 0 - odsyła ACK_FIGHT oraz dopisuje konstruktora do listy FightQueue w odpowiednie miejsce zależne od jego priorytetu. Jeżeli jest większa od 0, konstruktor i odejmuje 1 od tej wartości.
    iv. ACK_FIGHT: ignorowane
    v. REQ_OPPONENT_FOUND - usuwa z kolejki zarówno konstruktora nadawcę jak i konstruktora o identyfikatorze zawartym w żądaniu. Jeżeli konstruktora o identyfikatorze zawartym w żądaniu nie było w kolejce, dodaje wartość 1 do pola odpowiadającego temu konstruktorowi w wektorze FightBuffer.
    vi. ACK_OPPONENT_FOUND - ignoruje
    vii. REQ_RETURN: odsyła ACK_RETURN
    viii. ACK_RETURN: zwiększa zmienną AckCounterReturn. Gdy jest ona równa liczbie n - 1, części zostają zwrócone