

Laborationsrapport

S0006D

Laboration 1a

Pontus Stenlund

Ponste-5@student.ltu.se

Laboration_1a i SVN

2017-01-31

Innehåll

1. Problemspecifikation	1
2. Användarhandledning	2
2.1 Köra lösningen.....	2
3. Algoritmbeskrivning.....	3
4. Systembeskrivning	5
4.1 BaseGameEntity.....	5
4.2 Agent	5
4.3 EntityManager	6
4.4 MessageSender	6
4.5 State<entity_type>.....	7
4.6 FiniteStateMachine<entity_type>	7
4.7 Message	7
5. Lösningens begränsningar.....	8
5.1 Begränsningar	8
5.2 Hur kunde begränsningarna undvikas?	8
6. Diskussion.....	9
7. Testkörningar	10

1. Problemspecifikation

Uppgiften gick ut på att skapa fyra agenter för ett spel som genom inre representation av sina tillstånd och egenskaper kan beskrivas genom en FSM (Finite State Machine).

Världen där agenterna "lever" i ska visas i form av text eller grafiskt.

Karaktärerna ska kunna göra följande saker i sina "liv":

Arbeta i minst två olika platser och former.

Handla (t.ex. mat, böcker).

Sova.

Äta.

Dricka.

Umgås med vänner.

Agenterna ska även kunna skicka meddelanden till varandra för att umgås vid en given tidpunkt. Men om en agent t.ex. är för hungrig så skall den inte följa med.

Allt detta ska medföra att agenterna rör sig olika mellan olika tillstånden och utför de olika uppgifterna i sina "liv".

Jag anser att jag har uppnått lösningen för betyg 3.

2. Användarhandledning

Projektet finns på SVN i mappen som heter S0006D AI/Laboration_1a.

2.1 Köra lösningen

För att köra min lösning så går man in i main.cpp och skapar agenter genom att anropa **Agent(int id, char* agentName)** eller **Agent(int id, char* agentName, int workSal, int nightSal)**

Agent(int id, char* agentName) skapar en agent med id och namnet agentName.

Id används för att hålla reda på vilken agent som är vilken. agentName används för utskrivningarna i konsolen.

Agent(int id, char* agentName, int workSal, int nightSal) skapar en agent med id och namnet agentName, som ökar sin interna integer money med workSal/nightSal varje för varje Update().

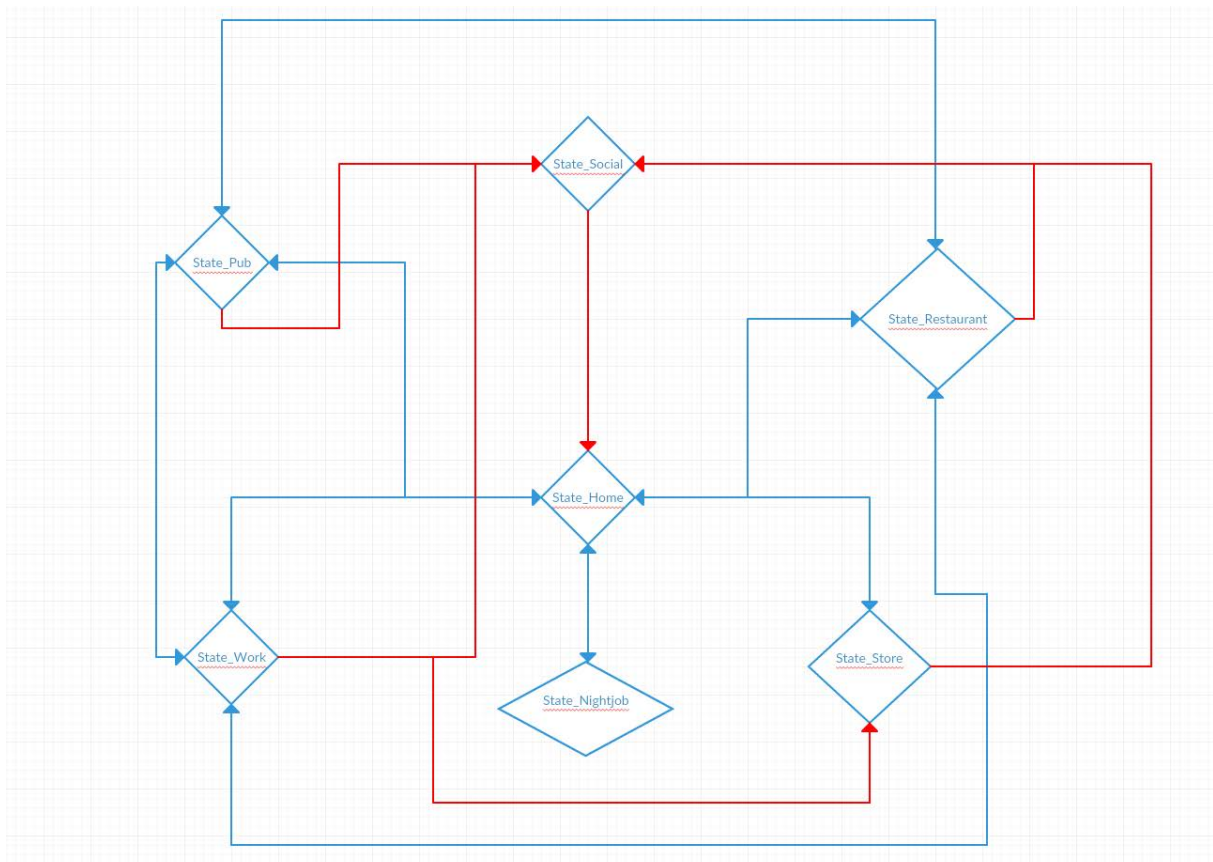
Man kan även sätta agenternas ökning på de interna värdena i varje Update() genom att anropa agent->SetGains(int hngr, int thirst, int fati, int soc).

I main.cpp så finns det en while loop. Där ska man anropa de olika agenternas Update().

Det sista inom scopet på loopen är att man ska skriva **Send->SendDelayedMessages()**.

Send->SendDelayedMessages() används för att skicka meddelanden som ligger i kön om tiden är inne.

3. Algoritmbeskrivning



Figur 1.State diagram

Här är ett diagram över de tillstånd som agenterna har. De blå strecken betyder att agenten kan fram och tillbaka mellan de tillstånd som den pekar på.

De röda strecken betyder att det är en enkel väg till det tillstånd som den pekar på. De flesta röda streck pekar på tillståndet social. Så de kan gå till social men från social kan de bara gå hem och sova.

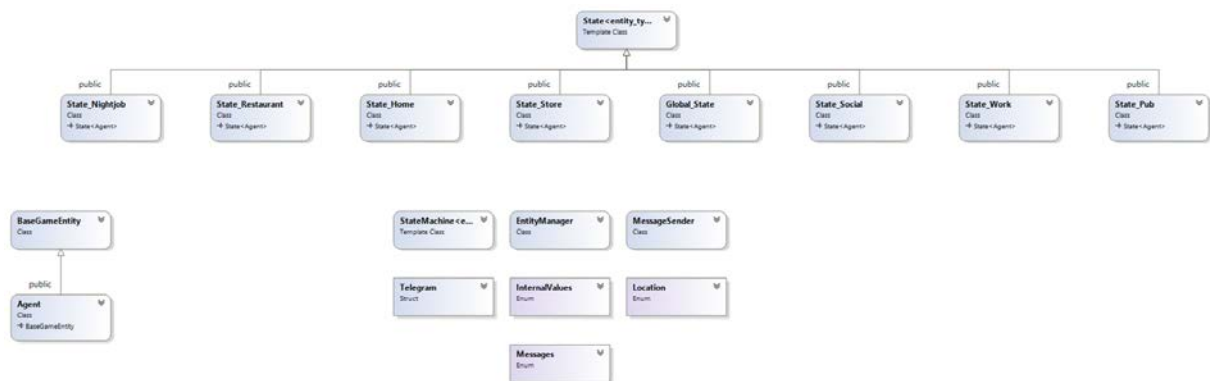
För att byta tillstånd så gör agenten följande:

1. Kolla om någon inre variabel är större än gränsvärdet i Update()
 - a. Kolla vilken variabel som är störst
 - i. Byt till det tillstånd som minskar den
2. Annars så kör vi det nuvarande tillståndet

För att byta till tillståndet State_Social så görs följande

1. Vi kollar i varje tillstånd ifall socialNeeds är stort nog.
 - a. Vi skickar ett msg_social till alla andra agenter.
 - b. Agenterna kollar ifall de är nog mätta och har nog mycket pengar.
 - i. De skickar tillbaka ett msg_confirmed tillbaka till avsändaren.
 - ii. De skickar ett försenat msg_timeToGo till sig själva.
2. Agenten som får ett msg_confirmed.
 - a. Har agenten inte har fått ett tidigare.
 - i. Skickar ett försenat msg_timeToGo till sig själva.
3. När agenterna får sina msg_timeToGo.
 - a. Byter de tillstånd till State_Social.

4. Systembeskrivning



Figur 2. Klassberoenden

Som visas i figur 2, så har alla tillstånd ett starkt beroende av `State<entity_type>` p.g.a. de ärver från den.

4.1 BaseGameEntity

Är en abstrakt klass som Agent ärver av.

Förutom konstruktorerna så finns ett par funktioner som Agent även har.

Update(), ska användas av Agent för att exekvera den kod för det tillstånd som agent är inne i.

MessageHandler(), kollar vilken tillstånd agenten är inne i och kör tillståndets `OnMessage(BaseGameEntity* agent, Message msg)`.

SetName(char* agentName), sätter namnet på agenten.

ID(), returnerar agentens id.

4.2 Agent

Agent metoder som behöver närmare förklaring:

Void SetInternalValues(InternalValues intval, val), plussar på val på något av de interna värdena dom sätts av intval. InternalValues är en enum som finns i Agent.h.

Void SetGains(int hngr, int thirst, int fati, int soc), är en frivillig metod som sätter hur mycket varje agents interna värden ska öka i varje tillstånd.

Void SetSocialMessage(bool val), sätter den privata boolen `sentSocialMessage` till val.

Void SetReceivedMessage(bool val), sätter den privata boolen `receivedConfirmedMessage` till val.

Void SetColor(int col), sätter vilken färg på texten som agenten skriver ut.

Void SetTextColor(int colors), sätter färgen på texten i konsolen som agenten skriver ut.

Int GetInternalValues(InternalValues intVal), returnerar det interna värde som man begär.

Int Gains(InternalValues inVal), returnerar sin ökning för det interna värde som man begär.

Int GetColor(), returnerar den färg man har satt texten till.

Bool GetSocialMessage(), returnerar boolen sentSocialMessage.

Bool GetReceivedMessage(), returner boolen receivedConfirmedMessage.

Void CheckNextState(Agent* agent), kollar vilket av sina interna värden som uppfyller begränsningen och byter tillstånd till tillståndet som passer.

Bool AcceptMessage(), kollar om agentens är nog "mätt" och har nog mycket pengar så den kan byta tillstånd till State_Social. Används i tillståndens OnMessage(Agent* agent, const Message& msg).

4.3 EntityManager

Är en singleton vars uppgift är att hålla reda på alla agenter som instansieras.

GetInstance(), returnerar pekaren till objektet.

RegisterEntity(int id, BaseGameEntity* newEntity), lägger in agenten i en map som innehåller alla agenter.

RemoveEntity(BaseGameEntity* entity), tar bort agenten från mappen.

GetAllEntities(BaseGameEntity* entity), returnerar en vector som innehåller alla agents id förutom den agenten som anropar funktionen.

4.4 MessageSender

Klassen är en singeton som hanterar alla meddelanden som skickas från agenterna.

GetInstance(), returnerar pekaren till objektet.

SendMessage(double delay, int sender, int receiver, int msg), skapar ett meddelande som skickas iväg direkt ifall delay är satt till noll. Är det satt till ett större tal än noll så läggs det in i en kö. Sender och receiver är till för att hålla reda på vilken agent som skickade och vilken agent som ska ta emot meddelandet. Msg är en enum som bestämmer vilken typ av meddelande det är.

SendDelayedMessage(), skickar meddelanden som finns i kön om tiden har gått ut.

4.5 State<entity_type>

State är en template abstrakt klass som alla andra State_<insert name here> ärver ifrån.

Alla State_<insert name here> är en singleton. Alla andra States innehåller följande metoder.

Enter(), kollar ifall agentens plats stämmer överens med den plats som tillståndet är satt till.

Execute(), körs hela tiden tills gränsvärdet är uppnått. Skriver ut en sträng så att användaren kan följa vad agenten gör.

Exit(), skriver ut i konsolen att agenten går ifrån tillståndet.

OnMessage(), körs när ett meddelande tas emot. Varje tillstånd skriver ut olika saker beroende på vilket tillståndet är.

4.6 FiniteStateMachine<entity_type>

Är en template singleton. klassen används för att hålla reda på och köra varje agents tillstånds Execute() och är tänkt för att man ska kunna gå tillbaka till det föregående tillståndet.

SetCurrentState(State<entity_type>* s), sätter currentState.

SetPreviousState(State<entity_type>* s), sätter previousState.

Update(), anropas från Agent för att köra agentens tillstånds Execute().

ChangeState(State<entity_type>* newState), byter agentens sätter previousState till currentState. Och currentState till newState. Anropar tillståndens Enter() och Exit().

RevertToPreviousState(), byter agentens tillstånd till previousState.

IsInState(const State<entity_type> st), kollar ifall agentens tillstånd är samma som st.

HandleMessage(const Message& msg), används för att kolla i agentens tillstånd ifall tillståndet hanterar meddelandet som agenten får.

4.7 Message

Är en struct som används som meddelande för att kommunicera mellan agenter.

MessageToString(), används för att kunna skriva ut till konsolen vilket meddelande som skickas.

5. Lösningens begränsningar

En begränsning som jag vet är att det är ganska ofta som alla agenter är i samma tillstånd.

5.1 Begränsningar

1. Agenterna är ofta i samma tillstånd dvs att det inte är helt balanserat.
2. När agenterna skickar meddelanden så skrivs mycket text ut så det blir svårt att följa.
3. Om ingen vill agent vill umgås så sätts den agenten som ville umgås sin socialNeed till 0. Detta är för att förhindra att den hela tiden ska skicka en förfrågan.

5.2 Hur kunde begränsningarna undvikas?

1. Börjat balansera tidigare.
2. Jag kunde t.ex. sätta in en paus efter varje utskrift i MessageSender.

6. Diskussion

Jag har stött på ett par problem.

Det största problemet jag stött på är att jag ville använda `<windows.h>` för att kunna pausa programmet en sekund så definierade den en av mina funktioner till något annat. Och när jag sedan skulle anropa den funktionen så hittade den inte åt den. Det tog ungefär fyra timmar innan jag hittade det felet och kunde lösa det.

Har också fått diverse linker errors p.ga. dålig planering när det gäller `#include` i mina `.h` filer.

Allt som allt så tycker jag att laborationen har varit väldigt givande och lärorik. Har varit väldigt roligt att få insikt i hur man kan utveckla en AI.

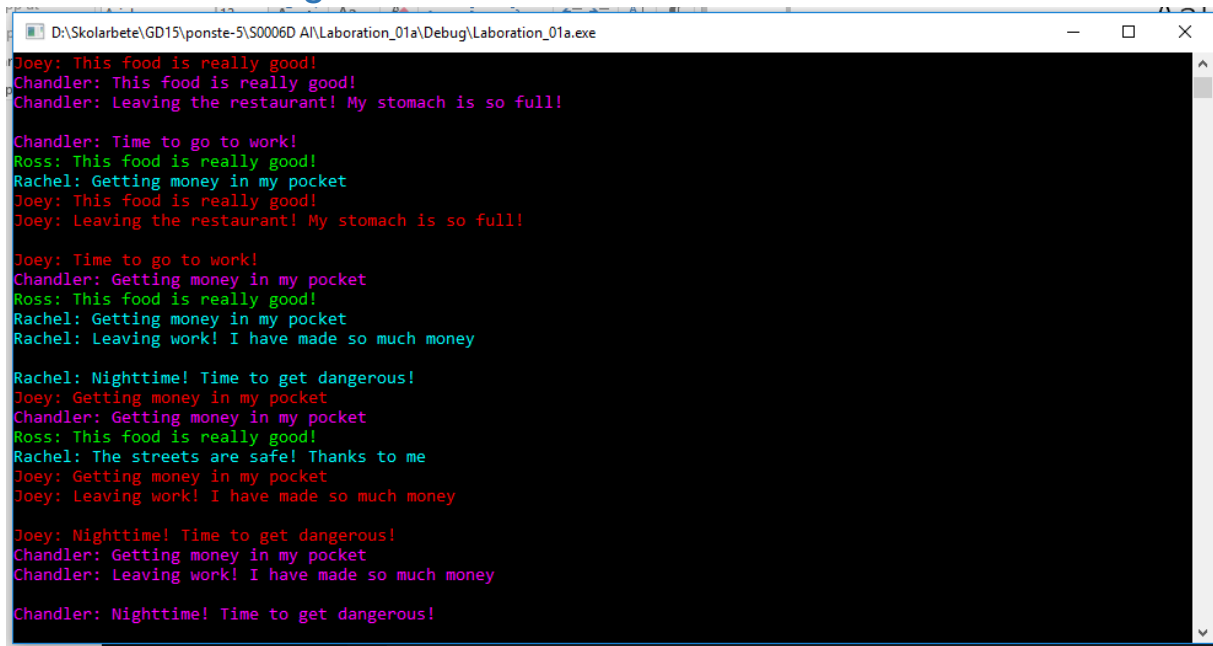
Men till nästa laboration ska jag ha en bättre planering. Har fått ändra ganska mycket för att jag har planerat för dåligt.

Jag har tagit information och inspiration från:

https://github.com/HEP85/game-ai/tree/master/Buckland_Chapter2-State%20Machines

Jag vill tacka alla mina klasskamrater som har gett mig goda råd och för alla givande diskussioner som vi har haft angående labben.

7. Testkörningar



```
D:\Skolarbete\GD15\ponste-5\S0006D AI\Laboration_01a\Debug\Laboration_01a.exe
Joey: This food is really good!
Chandler: This food is really good!
Chandler: Leaving the restaurant! My stomach is so full!

Chandler: Time to go to work!
Ross: This food is really good!
Rachel: Getting money in my pocket
Joey: This food is really good!
Joey: Leaving the restaurant! My stomach is so full!

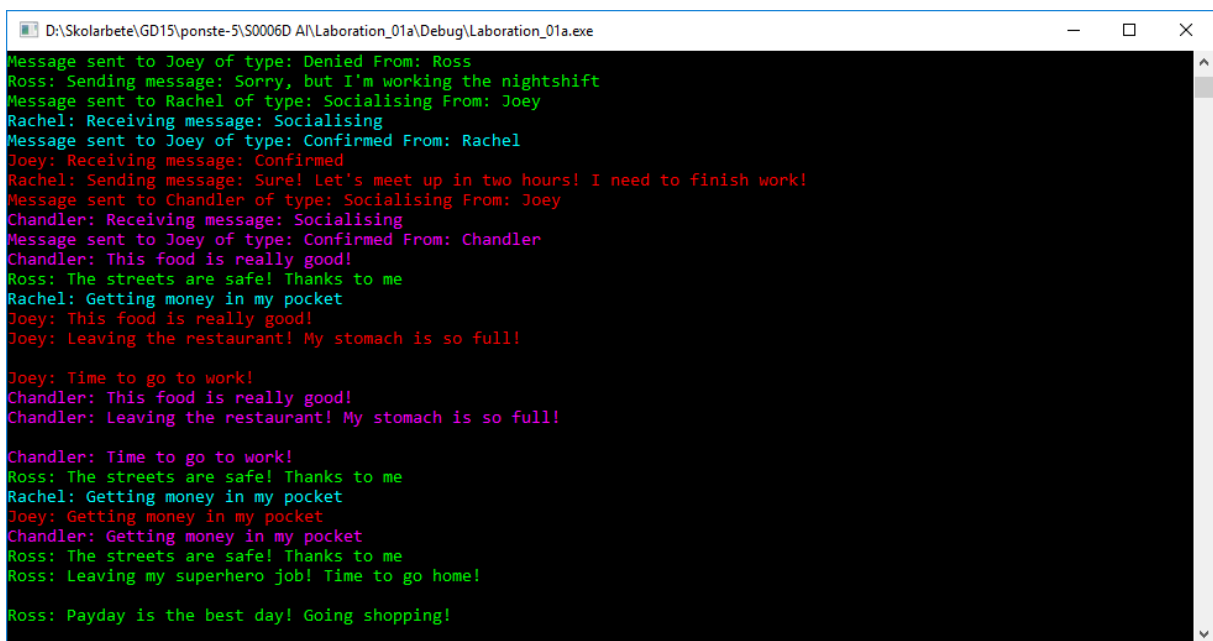
Joey: Time to go to work!
Chandler: Getting money in my pocket
Ross: This food is really good!
Rachel: Getting money in my pocket
Rachel: Leaving work! I have made so much money

Rachel: Nighttime! Time to get dangerous!
Joey: Getting money in my pocket
Chandler: Getting money in my pocket
Ross: This food is really good!
Rachel: The streets are safe! Thanks to me
Joey: Getting money in my pocket
Joey: Leaving work! I have made so much money

Joey: Nighttime! Time to get dangerous!
Chandler: Getting money in my pocket
Chandler: Leaving work! I have made so much money

Chandler: Nighttime! Time to get dangerous!
```

Figur 3. Agenterna i olika tillstånd



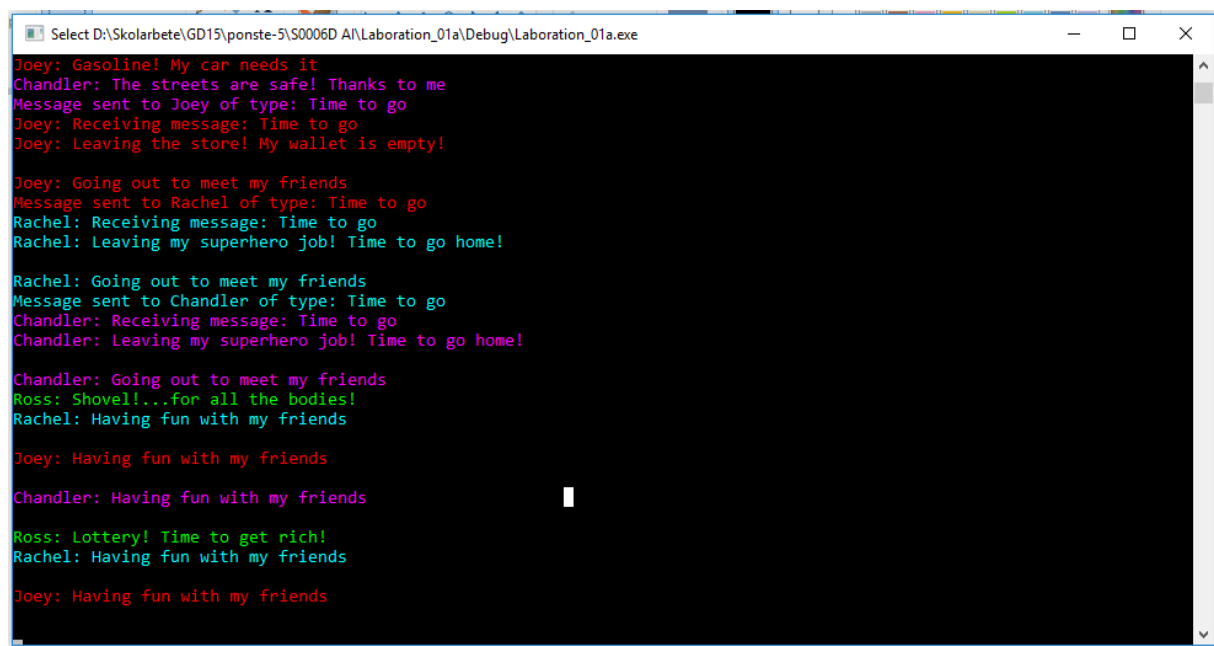
```
D:\Skolarbete\GD15\ponste-5\S0006D AI\Laboration_01a\Debug\Laboration_01a.exe
Message sent to Joey of type: Denied From: Ross
Ross: Sending message: Sorry, but I'm working the nightshift
Message sent to Rachel of type: Socialising From: Joey
Rachel: Receiving message: Socialising
Message sent to Joey of type: Confirmed From: Rachel
Joey: Receiving message: Confirmed
Rachel: Sending message: Sure! Let's meet up in two hours! I need to finish work!
Message sent to Chandler of type: Socialising From: Joey
Chandler: Receiving message: Socialising
Message sent to Joey of type: Confirmed From: Chandler
Chandler: This food is really good!
Ross: The streets are safe! Thanks to me
Rachel: Getting money in my pocket
Joey: This food is really good!
Joey: Leaving the restaurant! My stomach is so full!

Joey: Time to go to work!
Chandler: This food is really good!
Chandler: Leaving the restaurant! My stomach is so full!

Chandler: Time to go to work!
Ross: The streets are safe! Thanks to me
Rachel: Getting money in my pocket
Joey: Getting money in my pocket
Chandler: Getting money in my pocket
Ross: The streets are safe! Thanks to me
Ross: Leaving my superhero job! Time to go home!

Ross: Payday is the best day! Going shopping!
```

Figur 4. En av agenterna har skickat ett msg_social



```
Select D:\Skolarbete\GD15\ponste-5\S0006D AI\Laboration_01a\Debug\Laboration_01a.exe
Joey: Gasoline! My car needs it
Chandler: The streets are safe! Thanks to me
Message sent to Joey of type: Time to go
Joey: Receiving message: Time to go
Joey: Leaving the store! My wallet is empty!

Joey: Going out to meet my friends
Message sent to Rachel of type: Time to go
Rachel: Receiving message: Time to go
Rachel: Leaving my superhero job! Time to go home!

Rachel: Going out to meet my friends
Message sent to Chandler of type: Time to go
Chandler: Receiving message: Time to go
Chandler: Leaving my superhero job! Time to go home!

Chandler: Going out to meet my friends
Ross: Shovel!...for all the bodies!
Rachel: Having fun with my friends

Joey: Having fun with my friends
Chandler: Having fun with my friends

Ross: Lottery! Time to get rich!
Rachel: Having fun with my friends

Joey: Having fun with my friends
```

Figur 5. Agenterna har bytt tillstånd till social