Social Activities

Richard Evans, Thomas Barnet Lamb
revans@lionhead.com tblamb@lionhead.com

This paper is organised as follows. In section I, we argue a theoretical case for what we call activity-orientated structure. First, we argue that activities are an important part of human life, and one that will become increasingly important in future simulations including human-like agents in believable contexts. Second, we argue that since there are so many different activities, it is important they can be added easily to the system. Third, we argue a case for activities being realised in the game world as actual (albeit non-physical) objects, rather than (say) abstractions in the minds of agents.

In section II, we deal with implementational issues, focusing on the way activities should be added to the system. We argue that activities should be defined in a high-level language, which is compiled into C++. An example piece of activity-content, defined in this high-level language, is presented.

Section I – The case for activity-orientated structure

Why Social Activities are Important in the Simulation of Agents

Since this document is concerned almost exclusively with the notion of 'activities', it seems prudent to begin by trying to explain why we believe this concept merits such attention. In particular, we shall seek to establish successively that:

- Social activities are part of human life, and thus are usefully 'included' (in some sense) in simulations of human-like agents.
- Social activities are an important part of human life; thus their inclusion in simulations is correspondingly important to the believability of the simulation
- Social activities are the most important part of human life, that which distinguishes us from mere *brutes*. Thus, without addressing the activity concept in our simulations, we will be limited to simulating brutes*.*

Social activities are part of human life – agents who do not understand them can appear dumb

This minimal claim is most easily illustrated by considering a series of examples where an agent's lack of 'understanding' of activities and their relations can be cited as a failing of that agent.

- *A chess computer*. Although it is very good at chess, a chess computer is blind to the world around. It will never get bored of playing chess, and want to play something else. It does not understand, in other words, that playing chess is merely one activity amongst many, which satisfies some desires but not others. The chess computer doesn't understand the place of chess within the social flux – it doesn't understand that chess is a game played for recreation or competition.

- *A problem with Black & White.* In this game, a creature may be making friends with another, when he decides to interrupt this to go toilet. What is stupid about this behaviour is that the agent had no understanding of the consequences (for the activity of making friends) of stopping in mid-conversation to relieve himself. The same problem can arise in The Sims (a very entertaining piece of software): Peter's character was chatting up a lady, but then got tired, and in the middle of his chat he went off to have a bath. (The fact that this happens in both programs shows it is a moderately deep problem, and not a consequence of one particular implementation).
- *Agents' understanding of Ownership*. Nowadays, many games include agents owning objects. But this "ownership" is implemented in the simplest possible way. The following comments apply to SHRDLU, but apply directly to modern computer games. "SHRDLU cannot be said to understand the meaning of "own" in any but a sophistic sense. SHRDLU's test of whether something is owned is simply whether it is tagged "owned". There is no intensional test of ownership, hence SHRDLU knows what it owns, but doesn't understand what it is to own something. SHRDLU would understand what it meant to own a box if it could, say, test its ownership by recalling how it had gained possession of the box, or by checking its possession of a receipt in payment for it; could respond differently to requests to move a box it owned from requests to move one it didn't own; and, in general, could perform those tests and actions that are generally associated with the determination and exercise of ownership in our law and culture" [Herbert A Simon, "Artificial Intelligence Systems that Understand" (IJCAI-77, Proceedings) p.1064 [quoted in Dreyfus p.13] Agents (in current computer games) do not understand ownership because they do not understand the social activity in which ownership is embedded: ownership is a concept which belongs to the social activity of Enforcing Ownership, an activity which involves agents monitoring who owns what, and punishing others who mess with other people's things.

<u>Social Activities add *colour* to our lives, lives which are otherwise coarse and materialist</u>

From a certain perspective, our lives can seem empty. If all that there is in the world is other objects and agents, what is there for us to do except manipulate those objects and agents? All that we can want is to acquire as many objects as possible, and have as much influence over other people as possible.

This coarse and materialistic view of human nature is based on the assumption that all that exists is other objects and agents. Getting away from this dark picture involves admitting the existence of a variety of social activities. These activities elevate us because they give us new things to *want*. The lives of the Black and White creatures and the Sims characters are unquestionably materialistic because their *desires* are materialistic.

<u>Maximally strong claim: participation in sophisticated social activities is the critical property which distinguishes people from animals</u>.

Heidegger distinguished between merely existing, as an animal exists, and full-fledged human existence. (Heidegger called this "Dasein" or "being-there", and characterised it as participation in the social flux of the world). This insight has filtered through into popular culture: a well-known mobile phone company has mooted that "life is made of one to ones". Even Swedish popsters Abba echo the sentiment - "Without a song or a dance what are we?" – in other words, it is our participation in *forms of life* that makes us uniquely human.

The sophisticated skills we prize about ourselves, our ability to reflect, communicate, care about others, are dependent on our ability to participate in various social activities. Social activities, rather than just being one of the many things the mind thinks about, are actually the things which make sophisticated thought possible. It is *because* we can participate in sophisticated social activities that we can have the sophisticated thoughts that we prize. Philosophers have given many examples of aspects of sophisticated personhood which are conditional on participating in social activities.

- *Understanding of language.* Wittgenstein's language-games are examples in which someone knows what something means because he understands the activity he is in. Investigations $2: "The language is meant to serve for communication between a builder A and an assistant B. A is building with building-stones: there are blocks, pillars, slabs and beams. B has to pass the stones, and that in the order in which A needs them. For this purpose they use a language consisting of the words "block", "pillar", "slab", "beam". A calls them out; -B brings the stone which he has learnt to bring at such-and-such a call. If someone understands what "pillar" means, he does so in virtue of understanding its role in the activity in which it is embedded.
- *Compassion towards others*. Wittgenstein asks "Why can a dog feel fear but not remorse? Would it be right to say "Because he can't talk"?" [Zettel $518] The reason a dog can't feel remorse is that he cannot participate in the moral community in such a way as to recognise or manifest remorse. It is because he cannot participate in this activity that he cannot enter into these feelings.
- *Feelings*

  "The concept of pain is characterised by its particular function in our life. Pain has *this* position in our life, has *these* connections". [Zettel $532]

  "The concept of pain is bound up not just with characteristic pain-behaviour in circumstances of injury or illness, but also with pity and commiseration, fear and anxiety, cruelty and mercy." [B&H Vol3 p.68]
- *Structure and Organisation of Knowledge.* "Heidegger pointed out that the outer horizon or background of cultural practices was the condition of the possibility of determining relevant facts and features and thus prerequisite for structuring the inner horizon" [Dreyfus 36] In other words, we must not represent facts in one big pool, but must organise them structured by which social activities they are relevant to. This is

the way for agents to get efficient access to the things which are relevant in a particular situation. “The basic insight dominates these discussions that the situation is organized from the start in terms of human needs and propensities which give the facts meaning” [Dreyfus p.262]

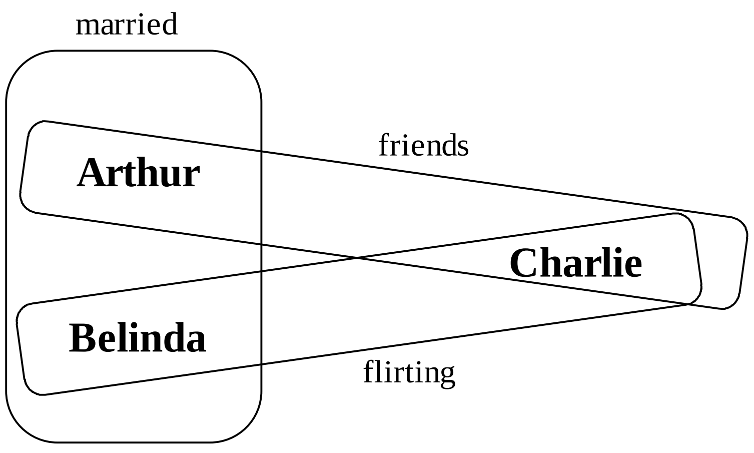
Activities are varied and must be easily addable

Here are a few paradigmatic examples of activities.

- Games (with winners, and without winners)
- Friendships
- Romantic attachments
- Conversations
- Marriages
- Households
- Social groups (e.g. the “in crowd”, “the nerds”)
- Meals
- Parties
- A night out; say to the cinema
- The moral community
- An insult
- Saying goodbye

Notice some of these activities, like a meal, are short-lived, but others are extremely long-term: a family lasts as long as there is anyone in it. Some activities can only exist within the context of a parent activity – a Goodbye activity, for example, cannot exist on its own. These activities which must have parents are called *sub-activities*.

These activities group people together, and these groupings can overlap:



In this dramatically charged example of activity-overlap, Arthur and Belinda are married. Arthur and Charlie are friends, but Belinda and Charlie are flirting.

Since there are so many different sorts of activity, we shouldn’t think in terms of “adding activities to our simulation” being a one-off operation. Rather, once our simulation has become activity-enabled, activities have become a kind of *content*. When building a simulation of a society of agents, there are many different levels at which we should easily be able to *add* new content – there is no magic formula or technique to generate plausible agents, there is just lots of different behaviour to simulate. Each bit of behaviour needs to be easily-addable; behaviour needs to be addable at lots of different levels (e.g. objects, animations, actions, and goals). If our simulations are to address activities, as we have argued they must, then activities become one of these levels. *The sophistication of a simulation is a function of the level at which we can easily add content*.

Should Activities Exist Inside or Outside the Minds of the Agents?

We have presented arguments aiming to illustrate that 'activities', in a broad sense, are something which must be present in realistic simulations of people. Further, we have argued that new activities should be easily addable. We have been deliberately vague, thus far, in precisely how this importance is to be addressed, and in what sense they should be 'included'. It is our claim that the best way to include them is for there to be actual 'activity-things', non-physical entities with their own state and internal logic to them, existing in the game world. Based on their state, they influence the behaviour of the agents within them, and based on what happens, they change their state. *Vitally importantly*, an 'activity-thing' does not *command* its participants to obey – it just *requests* that they perform an action, and tells them the consequences of ignoring this request. The agents themselves decide which requests to listen-to, based on their evaluation of the consequences of performing the various options.

There is one superficially impressive argument against this model, viz the fact that, in the real world, it seems that activities do *not* exist, at least in this sense. If they exist at all, they exist in our minds alone. "Ought we not, therefore, simply build the agent minds in an appropriate way, and hope that they 'discover' or 'learn' activities?"

The activities-really-exist approach has significant advantages over the all-in-agents'-minds one. Firstly, it is vastly simpler. Second, it is much less wasteful of memory – the state of activities would, in a simulation based on the second approach, have to be stored multiple times, once for each agent. Third, it is much clearer how newly created types of activity (the creation of which, we have argued, must be as simple as possible) will fit into the activities-really-exist approach – they just create new kinds of activity objects in the world. Under both approaches, the agents minds ultimately make the choices over which actions they want to take – activities are only a guide.

Having dismissed the in-agents-minds approach to activities, no other natural contenders, apart from the one we have suggested, present themselves. It has the virtue of straightforwardness, and of being (to some extent) tried-and tested: in *Black and White*,. towns, reactions and dances were all implemented as 'group minds'.

Now that we have explained why we think there should be 'activity-things' in the game-world, let us drop the shudder quotes and silly name, and call them simply *group activities*, or (where no confusion with the naive meaning of the word arises) simply *activities*.

Objection: “Are You Not Really a Behaviourist in Disguise?”

There is a worry that by focussing on the social activity, and allowing activities to influence how agents behave, we are taking something away from the individual agents – where is agent autonomy in this world of overlapping activities which jointly determine behaviour? Are we not really behaviourists in disguise, who are taking the cleverness out of the agent, and putting it into the social activities instead?

Remember that an agent does not *blindly follow* the commands of an activity. Instead, an activity *requests* that an agent does something (and communicates to him the consequences of failing to follow this request). An agent who is in many overlapping activities at once will have a number of requests pending at any time, and must choose between them. He decides which to follow by looking at the consequences of accepting or rejecting each request, and chooses the option which best satisfies his current goals.

“OK”, says the objector, “your agents have some autonomy because they can choose between the various requests they receive. But they are not *properly* autonomous because the range of choices is dictated from outside (by the activities), they do not themselves decide what the range of choices is”.

This objection misses the point made earlier, by Wittgenstein and others, that participation in an activity enables us to make choices we couldn’t otherwise make. Unless we are participating in the game of chess, we cannot choose between castling kingside and queenside. This choice is only available to us once we have entered into the chess-playing activity. Social-activities, in other words, are not constrictive but enabling. As another example, a couple cannot *get married* except in a societal and cultural context. Without culture, while they can visit a civic building, swap rings, say appropriate words, sign in a book and take lots of photos, they have not *got married* despite the fact that (in the particular culture and society that the authors come from) these are the physical actions that a couple do perform if they get married. Participation opens up new possibilities.

Section II – The implementation of activity-orientated structure

Requirements for an Implementation of Activities

Our activity implementation must satisfy these requirements

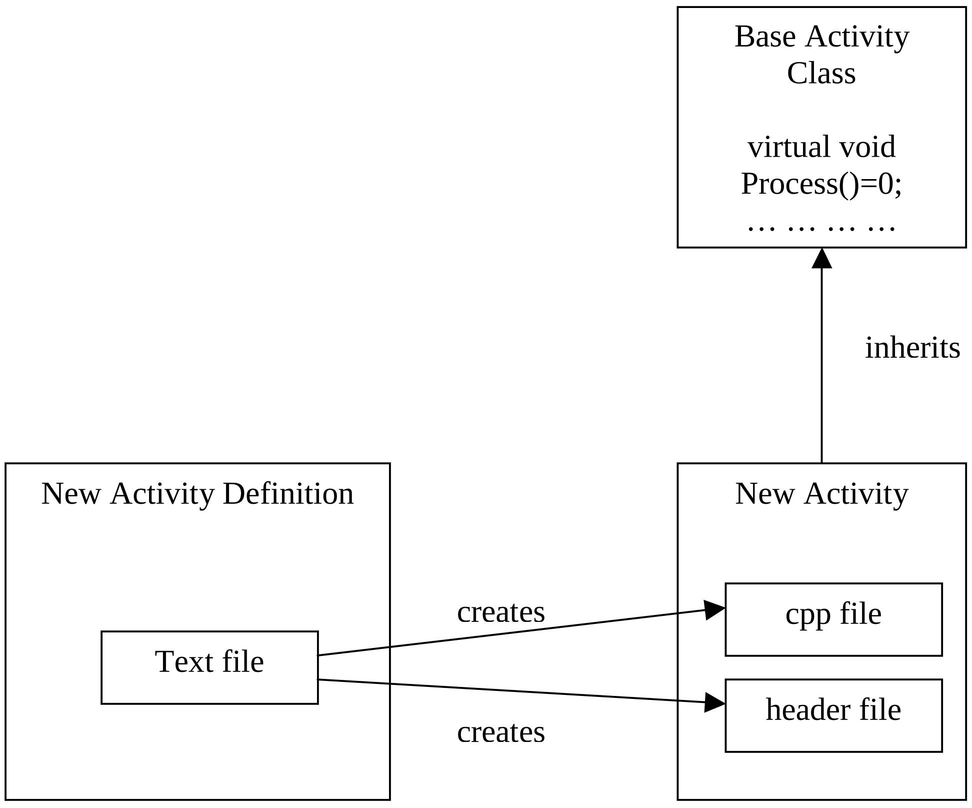
- It must be easy to add new activities
- Processing activities must be fast
- Activities must be easily debuggable
- The consequential structure of activities must be able to be queried by agents, so that they can determine whether or not it is in their interest to accede to requests.

Already these requirements seem almost incompatible: if activities must be efficient and debuggable, they should be written in C++ so we can use the existing optimisation and debugging tools, but if they are written in C++ it won't be very easy for content authors to add a new activity, because it will require a programmer, and it will be very difficult for the structure of the activity to be exposed to the agents: it might necessitate duplication of information, once to do things, and once to tell the agents what will be done. If, on the other hand, an activity is defined in a text file in a nice easy-to-use language, then content authors will be able to add new activities easily, and the agents will be able to use the information, but we lose the optimisation and debugging tools which C++ gives us. What to do?

The solution we have adopted is to define a new activity in a text-file, in a special-purpose easy-to-use language, but when this activity is parsed, c++ files are generated. This way we get the best of both worlds!

Implementation: Overview

A new activity is defined in a text file. When the activity is parsed, a new c++ activity is defined in cpp and header files. This activity inherits from the base Activity class:



The alternative to this might be some C preprocessor based technology (Game Programming Gems Vol 1, Section 3.1 – A Finite State Machine Class). This would have the advantages of being simpler (one would not need to write a parser) and one would gain the benefit that the code the debugger debugs would be precisely the same as that the user wrote, rather than merely generated from it. However, this would unacceptably straightjacket us – the preprocessor is a very weak tool. Moreover, writing a parser is not very hard using lex and yacc (note also that the hard part of writing a compiler from scratch, the semantics, can be partly avoided as we build our language on C++ and can leave some of the semantics to the C++ compiler). Moreover, on certain systems, via use of techniques like #line preprocessor directives, the debugger can be made to take the user directly to the appropriate line of the activity file, if the system breaks in an activity.

Activities in More Detail

Now we have decided that a dedicated ‘front’ to C++ as a form for an activity system, we must ask what form we would like that front to take. The first approximation to an answer would, for most game programmers, be a state machine. State machines have a proven history, and are very useful. However, it turns out to be useful to extend this slightly, to make the states hierarchical, and give states local variables. These are automatically constructed on entry and destroyed on exit. This is very similar to programming using the local stack machines in (Game Programming Gems 2, Section 3.2, Micro-Threads for Game Object AI) Thus an activity defines a state hierarchy:

```
activity Name(Parameter₁, … Parameter_N) {
        State1:
                State 1.1:
                        …
                        State 1.1.1
                                …
                        State 1.1.2
                                …
                State 1.2
                        State 1.2.1
        …
}
```

Each particular state can have its own parameters, its own preconditions, its own message handling, its own local variables with initialisations, its own actions, and its own sub-states.

Needs

Needs conditions specify what **needs** to be true for the activity to remain in this state, and what state to switch into if circumstances change. If we are in a particular state, all **needs** conditions in all parents states are checked before checking our own particular state's. If we have the state tree:

```
        S1:
                needs 1
                S2:
                {
                        needs 2
                        ..
                }
                S3:
                …
```

and we are in state S2, then needs1 will be checked, followed by needs2.

Requests

The only way an activity can affect the game-world is by issuing a **request** to an agent. Request statements ask agents if they wouldn't mind performing actions, and waits to find out if they actually bother to perform these activities.

A requests statement will appear in a state of the form:

```
        // needs conditions
        requests
                agent1.action1 !⇒ Rejection(1)
                →
                Finished
        {
                // actions to perform while waiting
        }
```

The first time this state is entered, the requests are passed to the agents concerned. From then on, we wait to see if any agent rejects the request; if so the rejection message is passed. If all agents finish their requested activities, state moves to Finished.

<u>Messages</u>

Messages are a way for activities to communicate without having to knowing each other's implementation details. Activities respond to messages using message-handlers. A message can have various parameters which are passed to the handler. If a message is passed to an activity, the handler function is called immediately. The handler may or may not change the state of the activity. Messages can be defined at any level of the state hierarchy – handlers lower down the chain take priority over more general handlers.

Example Activity

```
    ScissorsPaperStone(Agent p1, Agent p2, int max_score)
    {
        needs p1, p2
        constructor
        {
                int score1=0;
                int score2=0;
                int NumGoes=0;
                AGENT_LIST Players;
                enum actions
                {
                        scissors,
                        paper,
                        stone
                };
                Players.push_back(p1);
                Players.push_back(p2);
                -> Play
        }
        state Play
        {
                needs score1<max_score && score2<max_score !-> Finished
                requests
                        Players do CHOOSE(scissors, paper, stone) !-> GiveUp
                        ->
                        Evaluate
        }
        state Evaluate
        {
                if (p1.GetChoice() > p2.GetChoice())
                {
                        score1++;
                }
                else
                {
                        score2++;
                }
                -> Play
        }
        state Finished
        {
                if (score1 > score2)
                {
                        => UpdateScoreFromResult(p1) // send message to parent
                        -> WinnerCelebrates(p1, p2)
                }
                else
                {
                        => UpdateScoreFromResult(p2) // send message to parent
                        -> WinnerCelebrates(p2, p1)
                }
        }
        state WinnerCelebrates(Agent winner, Agent loser)
        {
                requests
                        winner do CELEBRATE() !-> GiveUp
                        ->
                        LoserCries(loser)
        }
        state LoserCries(Agent loser)
        {
                needs loser
                requests loser do CRY() -> GiveUp
        }
        state GiveUp
        {
                -> Delete
        }
    }
```

Conclusion

In this article, we have tried to motivate the introduction of *social activities* as the next obvious level at which to add content, and we have outlined a working system which makes it very easy to add a new activity (without having to worry about all the book-keeping needed to integrate our new activity with all the others). In our prototype, we already have a large number of activities running simultaneously: various games (both turn-taking games and games with simultaneous-turns), conversations, meal-times, courtship activities, with a moral community activity running in the background.

This work has been inspired by philosophers (Wittgenstein and Dreyfus) who are apparently critical of the very possibility of AI. It is pleasantly ironic that their work, which they might see as precluding the possibility of AI, will result in the next generation of social agents.

References

DeLoura (ed), Game Programming Gems, Volumes 1 and 2

Dreyfus, “What Computers Still Can’t Do”

Hacker, “Wittgenstein: Meaning and Mind”

Heidegger, “Being and Time”

Wittgenstein, “Blue & Brown Books”

Wittgenstein, “Philosophical Investigations”

Wittgenstein, “Zettel”