A Procedural Critique of Deontological Reasoning

Julian Togelius
Center for Computer Games Research
IT University of Copenhagen
Rued Langgaards Vej 7
2300 Copenhagen, Denmark
+45-7218-5277
julian@togelius.com

ABSTRACT
This paper describes a prototype game that learns its rules from the actions and commands of the player. This game can be seen as an implementation and procedural critique of Kant's categorical imperative, suggesting to the player that (1) the maxim of an action is in general underdetermined by the action and its context, so that an external observer will more often than not get the underlying maxim wrong, and that (2) most in-game actions are morally "wrong" in the sense that they do not contribute to well-balanced game design. But it can also be seen as an embryo for an authoring tool for game designers, where they can easily and fluidly prototype new game mechanics.

Keywords
Rules, mechanics, ethics, Kant, artificial intelligence, procedural content generation, procedural rhetoric

INDUCTION
Why are there rules, and why should anyone follow rules? In other words, what is the origin of rules, and from where is their legitimacy? These are question that game design has in common with ethics. A prominent attempt at justifying rule-based morality came with Immanuel Kant's (1785/1993) *categorical imperative*. Kant tried to base all of morality on logics, without recourse to individual or group utility (in stark contrast to the previously prevailing utilitarian doctrines). The most common formulation of the categorical imperative is its first formulation: "Act only according to that maxim whereby you can, at the same time, will that it should become a universal law." According to Kant, this makes certain actions immoral because the maxim they imply is contradictory. For example, stealing is wrong, because if you steal this implies the maxim "it is permissible to steal", which would dissolve the notion of property and make stealing impossible.

The categorical imperative has been very influential and is the base for most deontological approaches to ethics. But it can of course be criticized from many angles. For our purposes, one important question is: when you perform an act, exactly which maxim are you following? For example, a thief might claim to be following the maxim "it is permissible to steal, but only from rich people". Or, ”it is permissible to steal, but only on a Wednesday". Is the thief telling the truth, and if so, is this maxim contradictory? It might be argued that the thief himself would know the true maxim for his action; even if this were true (modern psychological research casts doubt over our ability to know our own motives (Wegner 2002)) the true maxim is not easily available to an outside observer. If the actor cannot be trusted to be perfectly truthful, an outside observer is tasked with inducing the correct maxim from one or several acts and the context of the act(s).

The problem is that there are generally many – perhaps infinitely many – possible interpretations of the same observations (e.g. acts, and the context of acts). This is a case of the problem of induction: how can we correctly infer a fact from a number of observations? Hume famously posed the problem, and doubted that it could at all be solved (1785/1939). Versions of the same problem are discussed in machine learning (Mitchell 1997, as "inductive bias") and in artificial intelligence (Dennett 1984, as the "frame problem").

But what has all this got to do with games?

At the core of any game is its system of rules. The rules, together with other game content and (in the case of digital games) the game engine are tasked with ensuring that the game is playable and interesting to the player. There is no magic formula for interesting games, but they are often well-balanced and have a long, smooth learning curve. As long as the player devotes time to the game and is immersed in it, the actions in the game carry meaning to the player, as do the achievements (as measured by e.g. "achievements") in the game. Conversely, a ruleset that fails to provide an interesting game experience – for example by being too easy or too hard – will make the player lose interest, and the actions and achievements in the game will cease to be meaningful.

Now, imagine a game where the player always acts according to a maxim that will become a universal law, because the game makes the maxims of the player's actions into universal laws. That is, the game finds the logic behind the player's actions and creates game rules out of them, which it then starts applying to the game. Would this game be an implementation of the categorical imperative?

It could be argued that it would. The original reasoning of Kant, that only actions for which the maxim could be universalized are morally "right" could even be seen as carrying over. Actions that, if universalized, create bad games would be "contradictory" because such actions would cease to have meaning within a non-interesting ruleset. As a simple example, if a player starts increasing his own score in response to simple actions that require no skill, this would remove the challenge in attaining high scores, and thus remove the meaning from the in-game score. An open question is to what extent this would extend to other, more complex actions and rules.

It could also be argued that implementing the categorical imperative in a game would be impossible, because the game does not know the maxim of the player's actions. But by the same reasoning deontological ethics itself would be impossible, except perhaps as a purely private undertaking, because we never know each other's real motivations and reasons (and quite possibly not our own). This argument (deontological ethics are futile because of the impossibility of induction) might or might not be correct, but it needs to be assessed empirically. Yes, empirically. Can we create a game system that correctly identifies the maxims behind a player's actions? If we manage to create a game that implements the categorical imperative as described above, the obvious question is what this would be good for. Will it make for a good game design? Will it help us teach or better understand the categorical imperative? Could such a game maybe help us teach ethics in general, or make game players more moral? Could an interactive exploration of the categorical imperative help us formulating a computational critique of deontological ethics? Before we can begin answering these questions we need to try to implement a game based on the categorical imperative.

The strategy that will be pursued in this paper is to build a system that generates game rules in response to player actions. Therefore, the exposition will first make a detour to procedural content generation, in particular generation of game rules, before proceeding to describe the system technically and then analyze it.

GAME RULE GENERATION

Several attempts have been made to construct systems that automatically design game rules. The motivating belief is that the computer can somehow help us design new games, either completely new games, or partial games based on existing rule fragments, themes or constraints. This could potentially help us both by expanding our limited human creativity (algorithms might not have the same preconceptions and taboos as we have), and by automating the tedious tasks of balancing and designing the details of mechanics. This could in turn be used to lessen the personnel costs involved in developing games and to allow non-experts to develop games. Several automatic game design systems have been implemented for research purposes, some based on evolutionary computation (Browne 2008, Togelius and Schmidhuber 2008), and others based on logical reasoning techniques (Nelson and Mateas 2007, Smith and Mateas 2008). Though each of these systems has shown some form of success, it’s fair to say that automatic game rule generation will remain an academic research topic for some time. One of the main problems is that in order to generate rulesets completely automatically, you also need to be able to evaluate them automatically. That is, algorithmically judge the quality of game rules. This is very much an open research problem.

Automatic ruleset generation is a form of procedural content generation (PCG), a more general term which refers to the automatic creation of game content of various types, such as levels, maps, characters and stories (Togelius et al. 2010). Not all PCG techniques aim at completely automated content creation. Recently, several attempts have been made to create mixed initiative systems, where a human designer and an algorithm take turns to edit the game content. One example is the Tanagra system for creating platform game levels, where a human designer can collaborate with a constraint solver to design the level; the human edits any part of the level at any time, and the constraint solver makes sure the surrounding parts fit what the human has created (Smith et al. 2010). Similar ideas can be found in systems for multi-level editing of landscapes, where algorithms take care of e.g. rearranging a city grid when a designer manually moves a river that flows through the city (Smelik et al. 2010). In a mixed initiative solution, the PCG algorithm thus becomes part of an authoring tool, where the human ultimately calls the shots.

Mixed-initiative rule generation

What would a mixed initiative PCG tool for game rules look like? Actually, “look like” is probably the wrong expression here, because rules don’t look like much. Beyond the most simple examples, reading a set of rules does not give an impression of what it is like to play according to that ruleset. Unlike landscapes and platform game levels, rules have an essential and irreducible “process dimension” or *procedurality*: they must be played to be grasped. Therefore, the model used in e.g. Tanagra, where a non-playing designer observes and manipulates the content from the perspective of an external observer cannot be used. Instead, the designer must be a player and build the game from within while playing it.

It is not entirely apparent how to construct such a tool. One idea could be to have the player/designer switch between construction phases, where rules are explicitly specified, and testing phases, when the rules are played. However, such an approach would chop up the playing experience in small slices for the player/designer, meaning that aspects of player experience that depend on a contiguous playing session might be overlooked; further, the need for the player/designer to explicitly specify the rules discounts the possibility for PCG to assist in inventing rules. Therefore, it would make more sense to equip the player with far-reaching abilities to modify the game while it is being played, and have the game system extract a consistent ruleset from the player’s actions.

The remainder of this paper describes a prototype system that does that. It will also be argued that this system implements the categorical imperative, and can therefore be used to provide a procedural critique of that philosophical idea, and ultimately the ethical systems that rest on it.

THE GAME

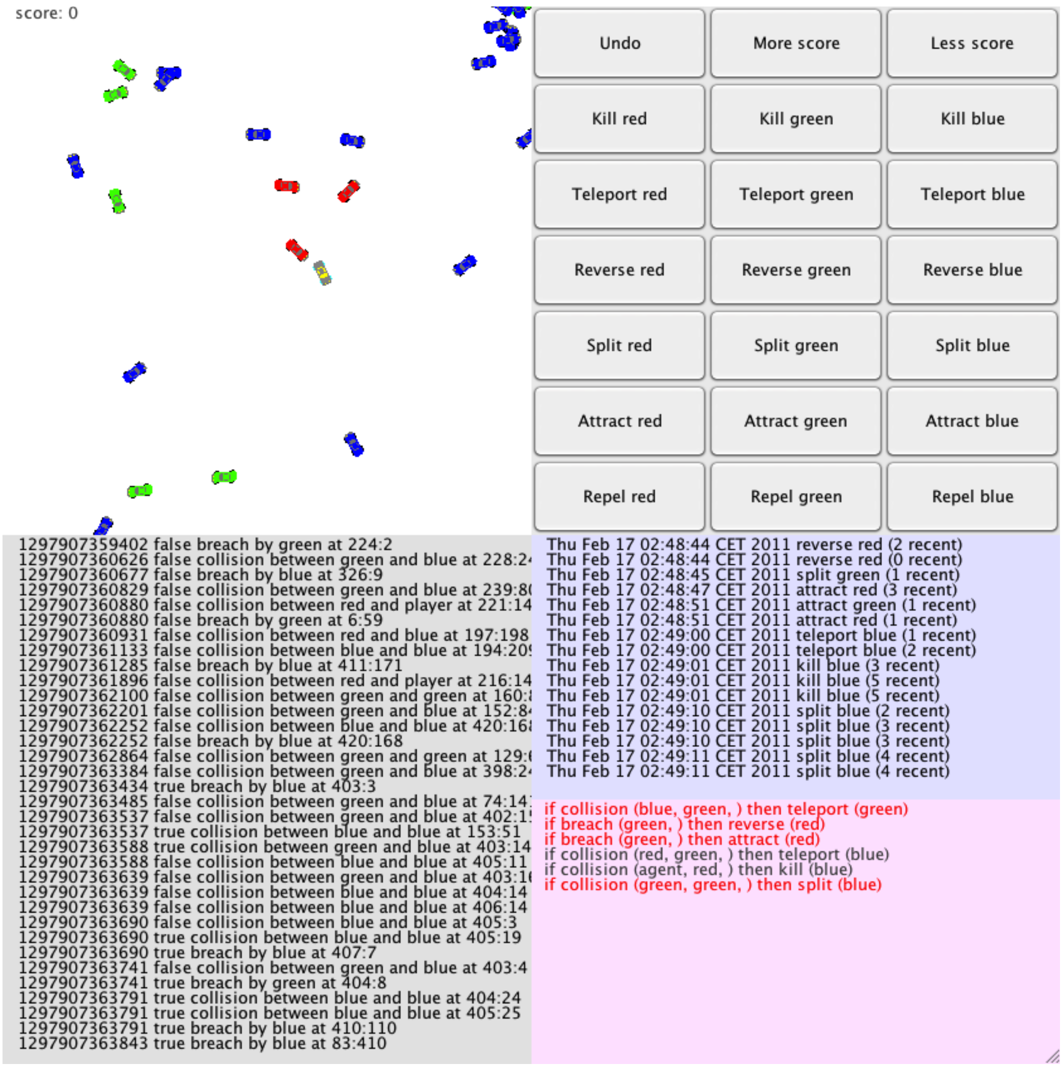
It would be untrue to say that the prototype game has no rules to begin with. (A good argument could be made that any digital game has some sorts of rules, as there exists some mapping between inputs (player actions) and outputs (e.g. visuals).) To begin with, the game has an ontology: there is a player-controlled agent, and several non-player characters, so called things. Each thing has one of three colours: red, green or blue (echoing the configuration in (Togelius and Schmidhuber 2008) and (Smith and Mateas 2010)), and a spawning mechanism ensures that there is always at least one thing of each colour in play. The game also features simple two-dimensional physics, where the player agent and the things individually possess not only a real-valued position but also an orientation and a velocity. The semiholonomic agents can accelerate and decelerate according to their orientation and to turn around, and they are affect by drag so as to limit maximum speed and drift.

The player controls the player agent through a keyboard interface, where he or she can choose to decelerate, accelerate or turn with the standard WASD keyset. The feel of the controls is somewhere between Asteroids and a simple racing game. Meanwhile, the things are controlled through simple heuristics: a target is selected, and they decelerate or accelerate so as to approach or avoid the target. Usually, each thing selects a random other thing as its new target every couple of seconds or so, and if the other thing is of a different colour than itself it chases it, otherwise it avoids it. Things can sometimes chase or avoid the player agent as well, as will be described below.

Many of these facts could well be described as rules, but they could also be described as technicalities relating to the game engine. It is up to the reader whether he or she prefers to call the movement speed in Halo or the physics of the dice in a board game rules; game designers Salen and Zimmermann (2004) would call such facts *constitutive* rules.

However, another kind of rules – the *operational* rules – are clearly missing, as events have no consequences.

Each time two things collide, or the player agent collides with a thing, or either some thing or the player agent reaches the end of the screen, an event is generated inside the game engine. There are two types of events: collisions, involving two things and a thing and the player agent, and breaches, involving a thing or an agent. Properties of an event also include its position and timestamp. As the game is played, a scrolling text window below the gameplay area displays the latest dozen or so events. When the game starts, nothing happens when an event is triggered. Things and the player character simply move straight through each other, and when they reach the border of the play area they simply stop as if they had run into a wall.



```
1297907359402 false breach by green at 224:2
1297907360626 false collision between green and blue at 228:2
1297907360677 false breach by blue at 326:9
1297907360829 false collision between green and blue at 239:8
1297907360880 false collision between red and player at 221:14
1297907360880 false breach by green at 6:59
1297907360931 false collision between red and blue at 197:198
1297907361133 false collision between blue and blue at 194:20
1297907361285 false breach by blue at 411:171
1297907361896 false collision between red and player at 216:14
1297907362100 false collision between green and green at 160:
1297907362201 false collision between green and blue at 152:8
1297907362252 false collision between blue and blue at 420:16
1297907362252 false breach by blue at 420:168
1297907362864 false collision between green and green at 129:
1297907363384 false collision between green and blue at 398:2
1297907363434 true breach by blue at 403:3
1297907363485 false collision between green and blue at 74:14
1297907363537 false collision between green and blue at 402:1
1297907363537 true collision between blue and blue at 153:51
1297907363588 true collision between green and blue at 403:14
1297907363588 false collision between blue and blue at 405:11
1297907363639 false collision between green and blue at 403:1
1297907363639 false collision between blue and blue at 404:14
1297907363639 false collision between blue and blue at 406:14
1297907363690 false collision between blue and blue at 405:3
1297907363690 true collision between blue and blue at 405:19
1297907363690 true breach by blue at 407:7
1297907363741 false collision between green and blue at 403:4
1297907363741 true breach by green at 404:8
1297907363791 true collision between blue and blue at 404:24
1297907363791 true collision between blue and blue at 405:25
1297907363791 true breach by blue at 410:110
1297907363843 true breach by blue at 83:410
```

```
Thu Feb 17 02:48:44 CET 2011 reverse red (2 recent)
Thu Feb 17 02:48:44 CET 2011 reverse red (0 recent)
Thu Feb 17 02:48:45 CET 2011 split green (1 recent)
Thu Feb 17 02:48:47 CET 2011 attract red (3 recent)
Thu Feb 17 02:48:51 CET 2011 attract green (1 recent)
Thu Feb 17 02:48:51 CET 2011 attract red (1 recent)
Thu Feb 17 02:49:00 CET 2011 teleport blue (1 recent)
Thu Feb 17 02:49:00 CET 2011 teleport blue (2 recent)
Thu Feb 17 02:49:01 CET 2011 kill blue (3 recent)
Thu Feb 17 02:49:01 CET 2011 kill blue (5 recent)
Thu Feb 17 02:49:01 CET 2011 kill blue (5 recent)
Thu Feb 17 02:49:10 CET 2011 split blue (2 recent)
Thu Feb 17 02:49:10 CET 2011 split blue (3 recent)
Thu Feb 17 02:49:10 CET 2011 split blue (3 recent)
Thu Feb 17 02:49:11 CET 2011 split blue (4 recent)
Thu Feb 17 02:49:11 CET 2011 split blue (4 recent)
```

```
if collision (blue, green, ) then teleport (green)
if breach (green, ) then reverse (red)
if breach (green, ) then attract (red)
if collision (red, green, ) then teleport (blue)
if collision (agent, red, ) then kill (blue)
if collision (green, green, ) then split (blue)
```

Figure 1: Screenshot of the game. The grey-and-yellow thing at the center of the play area is the player agent.

Figure 1 is a screenshot of the game. The game window is divided into four equal-sized quadrants. The top left quadrant is the play area, where the player agent and things move around. In the bottom left quadrant, recent events are displayed. The top right quadrant houses a matrix of buttons and is the command panel, from which the player can give commands to the game. Finally, the bottom right quadrant is divided into two text panels: the command panels, listing the most recent commands given by the players or the rules, and the rule panel, listing all of the rules of the game. When the game starts, both the event, command and rule panels are empty. We will now describe the functions of commands and rules.

Commands

The player can issue *commands* to the game by clicking on buttons on the command panels. Commands allow the player to control the game in a number of ways, giving him or her almost absolute control over the game. The following types of commands are implemented in the current version of the game:

- *Reward*: increase the score of the player by 1.
- *Punish*: decrease the score of the player by 1.
- *Kill*: remove a thing of the specified colour from play.
- *Teleport*: move a thing of the specified colour to a random place (not on top of any existing thing or the player) on the play area.
- *Reverse*: change the orientation and velocity of a thing of the specified colour to the opposite of what it was, and make it choose a new target.
- *Split*: duplicate a thing of the specified colour, and assign the copy a nearby position and opposite velocity of the copied thing.
- *Attract*: make all things of the specified colour choose the player agent as their new target, and move towards it.
- *Repel*: make all things of the specified colour choose the player agent as their new anti-target, and move away from it.

All command types except reward and punish take a colour as an additional argument, and there are therefore three buttons for each of those command types (Kill red, Kill green, Kill blue etc).

When the player chooses to issue a command that targets an individual thing – the kill, teleport, reverse and split command types – the game needs to figure out which particular thing of the specified colour the command applies to. If there is only one thing of the specified colour, the choice is trivial. If there are several, the game is faced with a non-trivial task of interpreting the player's will. The game must make some sort of assumption of causality here, e.g. that the reason for the command was that the thing was recently involved in some sort of event, such as a collision or a breach.

The current approach the game takes is to check all the recent events (recent events are in the current version of the game defined as those that happened within the last second) and see if there is one that includes a thing of the current colour. If there is at least one, it simply chooses the thing of the specified colour that was involved in the most recent event as the target of the command. If there are no recent events involving things of the specified colour, it chooses the thing of the specified colour that was closest to the most recent event. If there have not been any events at all recently, it simply chooses the thing of the specified colour that is closest to the player agent.

Now, one might argue that this interpretation problem is artificial – surely it must be possible for the player to specify exactly which thing he or she means to kill, split, teleport or reverse? Not necessarily. This would require the player to first click the button for the appropriate command, and then click the individual thing he or she would like to apply the command to. Given the pace of the game, the number of things typically on screen and the identical look of things of the same colour, the average mouse-and-keyboard handler would seldom be able to pick the appropriate target. (18 year old Korean StarCraft players might be able to do it.) Of course, it would be possible to pause the game, select the appropriate command and target thing, and unpause it. But this would mean shifting between a game-playing phase and a rule-construction phase, which is what we are seeking to avoid.

It might still be argued that the command interpretation problem is artificial, because the problem of selecting a target thing is a "mere user interface problem". But there is nothing "mere" about user interface problems.

So far, we have described 3.5 of the 4 quadrants. The bottom half of the lower right quadrant displays the cur- rent set of rules at all times. Each time step where a new command has been issued by the player, the ruleset is recalculated from the set of all commands issued by the player and events that has occurred in the game so far.

Rules

How are the game's rules represented, and how are they generated? This is the tricky part, and where the more serious interpretation/induction problem arises. Rules could have many formats, inducing the rules could be done in any of a number of ways, and the code has been written so as to allow any of a number of machine learning algorithms to be interfaced. The current implementation, however, uses a very simple rule format and an equally simple rule induction routine. The rules are of the format:

- *if event (colour 1, colour 2, player) then command (colour)*

For example:

- *if collision (red, blue) then attract (red)* or
- *if breach (agent) then reward ()*

Each time and event occurs, the rule interpreter is tasked with seeing if any of the rules in the ruleset apply to this event, and if so issue the relevant command. In the two examples above, the first rule "fires" whenever a blue and a red thing has collided, and issues the command that all red things should be attract to the player agent. The second rule fires when the player reaches the end of the playing area and issues a command to increase the score. The rule-issued commands are then interpreted in the same way as human-issued commands are, which is often not a problem but real interpretation issues might arise when e.g. a command has a target colour which was not involved in the triggering event. These interpretation issues would be very hard to avoid without radically redesigning the rule representation, as rules of this form can not specify more precisely which thing they apply to.

Rule induction works as follows: first, those commands that have been issued only once are filtered out. Then, the rule induction mechanism tries to find the event that motivated each command. This is done by collecting all the recent events for each time the command was issued (i.e. those events that occured within the last second before each issuing). The left hand side ("if" part) of the new rule is then simply selected as the most common event to precede the command.

PLAYING THE GAME

When starting the game, the player is presented with empty event, command and rule panels, and the playing area is occupied by a handful of things in random positions going in random directions. These things seem to be chasing and avoiding each other for no particular reason, sporadically forgetting what they were doing and doing something else instead. (The player might or might not feel an affinity to and/or relatedness with the things on the screen.) After a short time, maybe even within the first few seconds, the first events will begin to show up on the event pane, as some things happen to collide or bounce into the edges of the playing area. The player will probably test the waters a bit by moving the player agent around by using the WASD keys, and quickly figure out that (1) the control is quite nimble and the player agent is somewhat faster than the things and that (2) absolutely nothing happens when he/she drives into one of the things or the end of the screen, except that more events are generated.

Irresponsible play

What happens next is up to the player. A first-time player will maybe try the various command buttons one at a time and observe what happens. Well, increasing and decreasing the score is not a lot of fun in itself. The command types with individual thing targets (kill, teleport, reverse and split) do what they say, except that when one presses the button just to try it out it is hard to tell which thing of the specified colour that will suddenly disappear, change direction or duplicate. This is because as soon as there are more than three or so things on the screen, it is very hard for an observer to keep track of what's happening in general, and which the most recent event involving a thing of some colour was in particular. The attract and repel command types probably give the most immediate satisfaction, as their effect is so obvious: pressing all three attract buttons and suddenly having all things on screen coming towards you for a few seconds can actually be a bit disquietening.

As long as each button is only tried once, all is fine. But click the same button twice or more just to test, and the game will try to figure out the reason for this. As there was no particular reason for these commands being issued, the generated rule will be quite weird – per definition not as intended, as nothing was intended. What tends to happen very quickly here is that some command gets associated with a commonly occurring event with nothing to counter the effects of that command, making the game very unbalanced. The score could quickly run away to very low or high values, or the green things start teleporting around all the time. But the worst offender is the split command. Associate a split for things of some colour with an event involving things of the same colour, and you quickly get an out-of-control chain reaction, grinding the game engine to a halt as the screen fills up with things.

In other words, most non-deliberate actions are "immoral" within this system, as they quickly lead to an unbalanced and therefore meaningless game. Interpreting the system as an implementation of the categorical, this could be seen as a weakness of this imperative: an ethical system where most actions are immoral seems hard to reconcile with a meaningful degree of personal freedom. It should also be noted that the morality of any particular action depends on the other rules that are already implemented in the

Responsible play

But what happens when one sets out to deliberately design a playable game through issuing the appropriate commands in response to the appropriate events, which might or might not be created by the player through moving the agent? It turns out it can be done, but it is not very easy. A good strategy is to make an event have a short-term negative effect and a potential long-term positive effect. For example, colliding with a green thing makes a red thing split, but also attracts the blue things and teleports the green thing away. Colliding with a red thing kills the red thing and increases the score. Colliding with a blue thing decreases the score. This forces a gameplay where one has to take the risk of colliding with green things when the risk is small to be caught by the blue things, in order to later have the chance to "reap" the red things. However, constructing even this simple ruleset is a challenge, as the number of events happening means that the game frequently misinterprets your actions.

So, at least in principle, the system is a working prototype of a mixed initiative rule design/generation system, meant for improvising rulesets. The vocabulary of commands is quite rich, meaning that a large number of interesting rule- sets should be possible to represent. However, the system suffers from the problem of induction, leading to that rules other than those intended are often created. But then, if one knew exactly what rule one wanted to create, one could just add it to the ruleset manually.

The difficulty with making the system interpret the real reason for our commands might also point to another form for a possible procedural critique of the categorical imperative: it is computationally too difficult to induce the maxim for some action for the categorical imperative to work. (There is no reason why computational complexity considerations should not apply to philosophical arguments, given the decidedly limited computational capacity of the human mind (as opposed to the human brain).)

DISCUSSION

This paper describes a prototype system that can not reasonably be regarded as finished in any way other than that it actually runs without crashing. Much more work would remain to be done in order to make this a viable authoring tool, or game. Also, more work, including empirical studies of player behaviour, would be necessary in order to conclusively argue the effectiveness of the system as a tool for analysis and critique of philosophical ideas.

Induction

The rule induction mechanism could, and should, be swapped for something more sophisticated. The current mechanism only allows one rule for each command, only allows one event as a trigger for that command, and the method for select- ing the appropriate command is simplistic and atomic. As the problem of rule induction can easily be cast as a classification problem, any of a large number of well-developed algorithms from machine learning and computational intelligence could be used instead. In particular, standard decision tree learning algorithms such as C4.5 could be tried, or learning classifier systems. These algorithms can produce more complex rules based on an assessment of whole sets of events and commands.

Still, standard supervised learning algorithms are inductive, and will suffer from the problem of underdetermination of rules based on previous events. One of the most influential attempts at solving the problem of induction is Karl Popper's (1934/1959) critical rationalism or, as it is often called, falsificationism. The central concept here is one of trial and error: a hypothesis is advanced and then every effort is made to try to falsify the hypothesis by finding conflicting evidence. One could imagine a rule construction system that invents new rules that it knows will have an effect on the game as it is being played out, and allows the player/designer to falsify rules by explicitly disapproving of the unwanted ones (or their consequences).

Gameness

A purist might object to the current game that it is not a game at all: there is no way of winning, no way of losing, and no notion of progress. The only thing there is is a score counter, and as it is very simple to increase the score arbitrarily much this does not meet the minimum challenge requirement of a game. It is up to the reader to agree or disagree with this objection – there are many definitions of games. However, the system could easily be turned into something everybody would recognize as a game by gamifying it, or adding a game layer on top of it. For example, one might divide each level of the game into two phases: one where the player can issue commands, and one where commands are not available and the player can only interact by moving the player agent. The player can at any time go from the first mode to the second, but not back again before the end of the level. Each level would then have a goal that needs to be fulfilled to pass the level, e.g. to first earn 10 points and then lose them again, or to cause exactly 13 collisions between each pair of things without any one of them touching the border of the screen. The challenge would lie in the combination of inventing a ruleset, constructing it, and playing according to it so as to achieve the goal. The gameplay in such a game would have some similarities to training-based games such as NERO (Stanley et al. 2005) or Black and White (Lionhead 2001).

Procedurality

Bogost (2007) discusses how games can be used to persuade and critique, in particular through embodying an argument within a procedure. The game system models some entity or system outside of the game, and embodies certain assumptions and mechanics so that as the player plays the game he/she "completes" an argument about the entity or system that the game is modelling. Bogost describes examples from many different domains, including serious games about politics (e.g. modelling the economics of the healthcare industry to make an argument about the need for curtailing malpractice lawsuits) and the ideological framing in mass-market entertainment-focused games, such as GTA: San Andreas (Rockstar Games 2004) where the selection of available activities and consequences leads players to act according to particular values even though the open world of the game suggests that anything is possible.

The question is whether the system described here could be said to be a relevant representation of the categorical imperative (and in general of deontological reasoning building on it), and whether it can provide an interesting critique of this philosophical concept. This paper proposes to answer both questions in the affirmative. The game is a relevant representation of the categorical imperative because it puts the player in a situation where the player can do what he/she wants to (within the bounds of the system), but where the maxims of the player's own actions is forcefully enacted by the system. The game thus replicates (models) what can be see as the core logic of the categorical imperative. Further, the game can provide an interesting critique of the concept, because the player "completes" an argument by playing the game and noting the effects on the system from his/her own gameplay. In particular, the following two points can be seen as being expressed through the procedural rhetoric of the game:

- The maxim of any action, or sequence of actions, is indeterminate – the system frequently assumes you are acting according to a different maxim than youthought you were, or maybe assumed you were acting according to some maxim when you were not.
- Being "moral" is very hard, perhaps so hard as to make it unrealistic to ever be moral according to the categorical imperative – most actions that are not taken with extreme care end up producing a terrible ruleset which is not engaging to play with, thus voiding in-game actions of meaning.

Several objections could be raised against these arguments. For example, it could be argued that the inadequacy of the inferred maxims is simply due to the primitive machine learning system, that the vocabulary of events and commands is biased against meaningful game rules, or that the agents, events and commands are too removed from human reality to ever be meaningful. Such objections could be met with theoretical counterarguments (for example invoking the mathematical bounds on learning systems), but it would probably be more effective to build new and better systems using a similar rhetoric to prove the contrary point.

Relation to Sicart's framework

Sicart (2009) presents a framework for discussing and assessing the ethics of computer games. His framework is based on virtue ethics and information ethics rather than deontological ethics, and assigns much importance to the player community and the moral reasoning capacities of the player. The current system has no player community at all, and the individual games created even less so (indeed, games with procedurally generated rules might be the only truly single-player games in the world) and the account in this paper has not focused on the moral reasoning capacities of the player. Thus, it might seen that Sicart talks about game ethics from a quite different perspective than this paper does, and that there is negligible overlap in scope and methods. However, the argument in this paper agrees with Sicart that "bad design, then, is to be considered unethical" because "the game creates ludic experiences that may be harmful for the player as a moral being" (ibid, p. 144). According to Sicart, lack of balance is a typical example of bad game design. As we have seen, non-deliberate commands and actions in the Rulearn system easily creates unbalanced games.

Representation

Finally, it could be noted that in order for the actions in the game to be interpreted as having a moral value we would probably need to move from nondescript things bumping into each other in a 2D plane, to something players would be prepared to think of as characters. The design and technology ideas in this paper might for example form a good basis for a Sims-like game aimed at exploring ethical questions.

The roads ahead

As has been stated several times in the paper, the software described here is not a finished product. Additionally, the various arguments might doubtlessly be refined further. More work is needed. However, there a number of different paths for this work to take discussed in the paper, and those are to some extent exclusive (more work on the authoring tool aspect might not advance work on the metagame aspect, the procedural argument or the machine learning problem). Due to the woefully inadequate number of hours in the day, the author will not be able to pursue all the paths outlined above. It was therefore felt that this work should be published already in its current state, while the ideas are still fresh and hopefully able to inspire.

ACKNOWLEDGMENTS

Thanks to Mark Nelson, Ian Bogost, the anonymous reviewers and my dear colleagues at the Center for Computer Games Research for insightful comments. This research was funded by the Danish Research Agency (FTP) project

BIBLIOGRAPHY

I. Bogost. Persuasive Games. MIT Press, 2007.
C. Browne. Automatic generation and evaluation of recombination games. PhD thesis, Queensland University of Technology, 2008.
D. Dennett. Cognitive wheels: The frame problem of AI. In C. Hookway, editor, Minds, Machines and Evolution. Cambridge University Press, 1984.
D. Hume. A Treatise of Human Nature. Penguin Classics, 1985 (1739).
I. Kant. Grounding for the Metaphysics of Morals. Hackett, 1993 (1785).
T. Mitchell. Machine Learning. McGraw-Hill, 1997.
M. Nelson and M. Mateas. Towards automated gamedesign. In Proceedings of Artificial Intelligence and Human-Oriented Computing, 2007.
K. Popper. The Logic of Scientific Discovery.Routledge, 1959 (1934).
K. Salen and E. Zimmerman. Rules of Play: GameDesign Fundamentals. MIT Press, 2004.
M. Sicart. The Ethics of Computer Games. MIT Press, 2009.
R. M. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra. Integrating procedural generation and manual editing of virtual worlds. In Proceedings of the ACM Foundations of Digital Games. ACM Press, June 2010.

A. M. Smith and M. Mateas. Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games. In Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG), 2010.
G. Smith, J. Whitehead, and M. Mateas. Tanagra: A mixed-initiative level design tool. In Proceedings of the International Conference on the Foundations of Digital Games, 2010.
K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Real-time neuroevolution in the nero video game. IEEE Transactions on Evolutionary Computation, 9(6):653–668, 2005.
J. Togelius and J. Schmidhuber. An experiment in automatic game design. In Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG), 2008.
J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. Search-based procedural content generation. In Proceedings of EvoApplications, volume 6024. Springer LNCS, 2010.
D. M. Wegner. The Illusion of Conscious Will. MIT Press, 2002.