Goldsmiths
UNIVERSITY OF LONDON

Computational Creativity Group

Computational Creativity Autumn School II

Some Guiding Principles

Simon Colton

Computational Creativity Group
Department of Computing
Goldsmiths College, University of London

ccg.doc.gold.ac.uk

Let's Take a Crack at...

- The humanity gap
- Managing the perception that people have about the creativity (or lack thereof) in software
- Assessing progress in Computational Creativity research in terms of what software actually does

The Humanity Gap

Introduction

Mainstream poetry is a particularly human endeavour: written by people, to be read by people, and often about people. Therefore – while there are some exceptions – audiences expect the opportunity to connect on an intellectual and/or emotional level with a person, which is often the author. Even when the connection is made with characters portrayed in the poem, the expectation is that the characters have been written from a human author's perspective. In the absence of information about an author, there is a default, often romantic, impression of a poet which can be relied upon to provide sufficient context to appreciate the humanity behind a poem. Using such an explicit, default or romantic context to enhance one's understanding of a poem is very much part of the poetry reading experience, and should not be discounted.

The Humanity Gap

Software isn't human

People often want the human connection when they consume creations

There is a default assumption about non-creativity in software and a vicious circle

When people evaluate artefacts, they are often really evaluating the creative act which produced it, in particular the processes involved

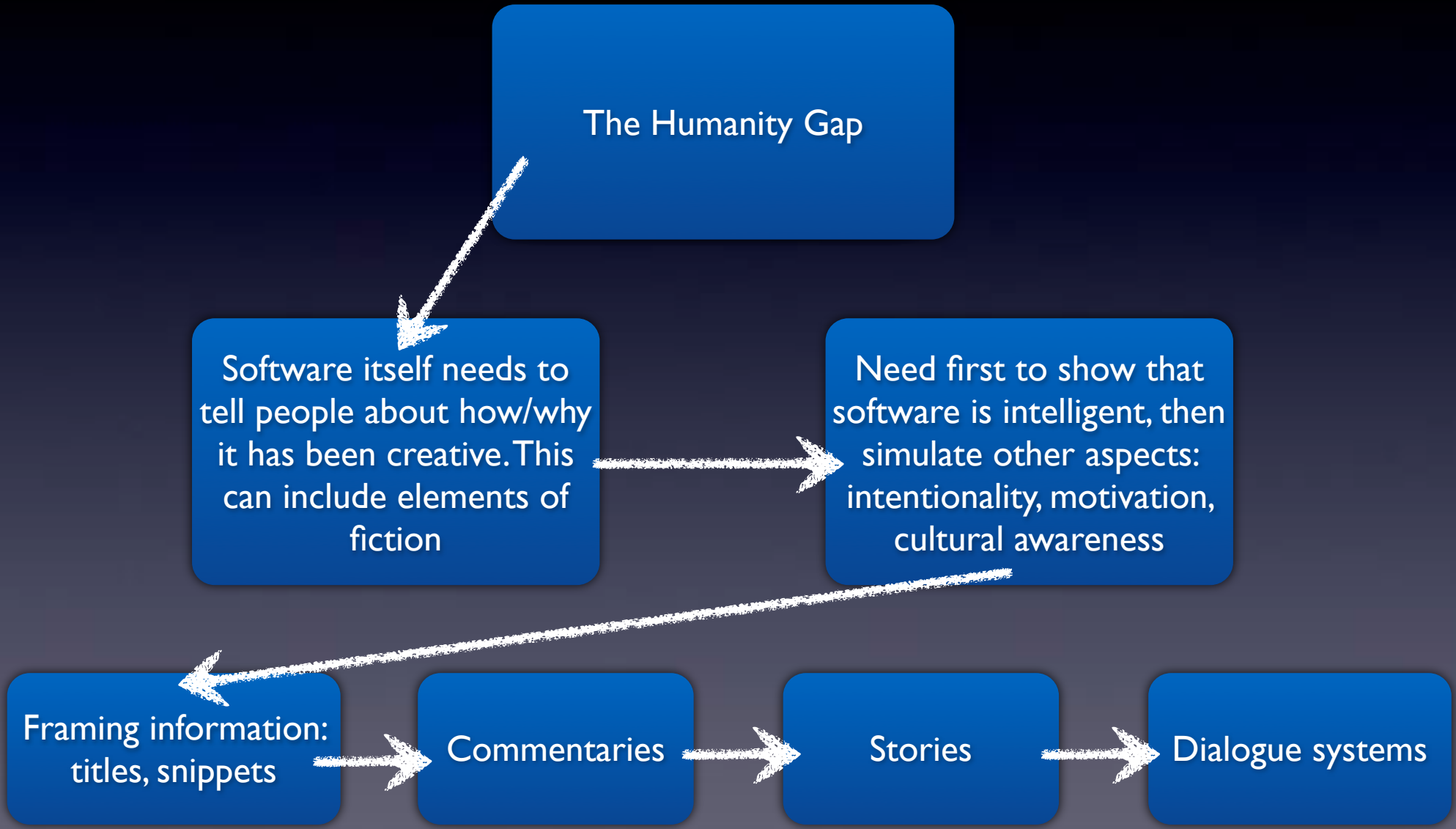
Created artefacts are invitations to a dialogue

To add drama, creative people often introduce elements of fiction into their explanations

Some mostly factual statements...

Filling the Humanity Gap #1

Showing Intelligence



Filling the Humanity Gap #2

Admitting Differences

- If we can admit that software and people are different creative beings, and that how artefacts are produced and by whom/what are integral part of the artefacts...
- ...then we should acknowledge computer generated artefacts as being fundamentally different to those produced by people...
- ...and we should drop any pretence that it's useful to compare human and computer generated artefacts
- ...so that we can concentrate on assessing progress in our research and exploring the possibilities for non-human creative behaviour

Filling the Humanity Gap #3

Adding Sophistication

- Appreciation
  - Need to have much more sophisticated models of how created artefacts might affect audiences intellectually and emotionally
- Imagination
  - Need to move from rough approximations blurring the lines of intentionality to full idea generation
- See the creativity spider later

Managing Perceptions

Avoiding the 'Uncreative' label

- 'Creativity' is an essentially contested, secondary concept which confuses people, processes and artefacts. We won't be agreeing about what 'creative software' is any time soon
  - But there is much more of a consensus about when software has been uncreative in its actions or lack thereof
- So, a more practical approach to building software is to address as many reasons as possible why people will call the software uncreative
  - This has been the approach with The Painting Fool project
- "We hope that one day, people will call The Painting Fool 'creative' because they can no longer think of a good argument why it is uncreative."

The Creativity Spider

- Your software can't be creative because...
  - It's not particularly skilful
  - It has no appreciation
  - It has no imagination
  - It doesn't exhibit any intentionality
  - It never learns for itself
  - It never reflects on what it has done and produced
  - It doesn't come up with any ideas
  - It's not particularly innovative

Write software which exhibits behaviours that can be called

- Skilful
- Appreciative
- Imaginative
- Intentional
- Adaptive(?)
- Reflective
- Inventive
- Innovative

Skilful

Appreciative

Imaginative

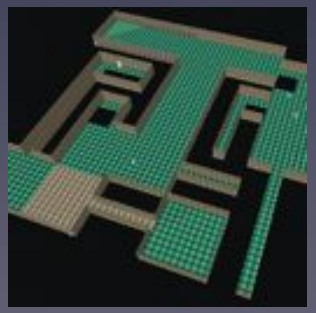
Intentional

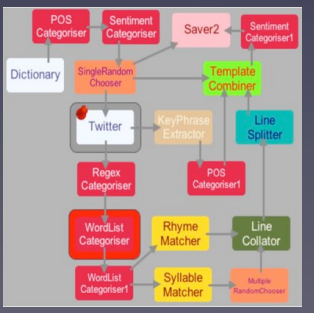
Adaptive(?)

Reflective

Inventive

Innovative





Assessing Progress

Computational Creativity Theory: The FACE and IDEA Descriptive Models

Simon Colton[1], **John Charnley**[1] **and Alison Pease**[2]

[1] Computational Creativity Group, Department of Computing, Imperial College, London, UK. ccg.doc.ic.ac.uk

[2] School of Informatics, University of Edinburgh, UK.

Abstract

We introduce computational creativity theory (CCT) as an analogue in computational creativity research to computational learning theory in machine learning. In its current draft, CCT comprises the FACE descriptive model of creative acts as tuples of generative acts, and the IDEA descriptive model of the impact such creative acts may have. To introduce these, we simplify various assumptions about software development, background material given to software, how creative acts are performed by computer, and how audiences consume the results. We use the two descriptive models to perform two comparisons studies, firstly for mathematical discovery software, and secondly for visual art generating programs. We conclude by discussing possible additions, improvements and refinements to CCT.

Introduction

Machine learning is a very well established research area, and has possibly had the greatest impact of all AI subfields, with successful applications to classification and prediction tasks in hundreds of areas of discourse. Machine learning researchers asked: "What does it mean to say that a computer has *learned* something?" In computational creativity research, we have similarly asked: "What does it mean to say that a computer has *created* something?" It is therefore sensible to look at how machine learning developed to suggest future directions for computational creativity research. Human learning involves many behaviours, including memorisation, comprehension, abstraction and generalisation. While textbooks like (Mitchell 1997) use general terms such as "learning how to do something by observing it being done", in reality, machine learning researchers have

putational learning theory (CLT). In (Ang
Anghain states that the goals of CLT are to:

*Give a rigorous, computationally de
plausible account of how learning ca*

Each word in this statement has been care
particular, by requiring *computational* desc
lights that CLT discusses actual algorithms
requiring *rigour*, he stresses that the purp
state and prove theorems about the nature
algorithms. Such theorems deal with learn
PAC model (Valiant 1984), i.e., which type
be induced by which methods, and with e
extent to which certain methods must sea
mentioning the *plausibility* of the account
grounds CLT to describing machine learni
with reasonable computational resources, b
nation and interaction requirements. Fina
the account to one which describes how
alone, Anghain emphasises that CLT does n
encompass approaches to computer learni
fers a particular formalism which research
ness progress. The benefits of CLT have be
via theoretical results being turned into pra
most notably the introduction of ensemble
such as boosting (Schapire 1990), whereby
tors are assembled from multiple weak one

We introduce here the first draft of a
computational creativity theory (CCT). In t
aim for CCT to provide a rigorous, computa
and plausible account of how creation can
step has been to provide two plausible de
which can be used to describe the processe

Computational Creativity Theory: Inspirations behind the FACE and IDEA descriptive models

Alison Pease[1] **and Simon Colton**[2]

[1] School of Informatics, University of Edinburgh, UK.

[2]Computational Creativity Group, Department of Computing, Imperial College, London, UK. ccg.doc.ic.ac.uk

Abstract

We introduce two descriptive models for evaluating creative software; the FACE model, which describes creative acts performed by software in terms of tuples of generative acts, and the IDEA model, which describes how such creative acts can have an impact upon an audience, given information about background knowledge and the software development process. We show how these models have been inspired both by ideas in the psychology of creativity and by an analysis of acts of human creativity.

Introduction

To enable the Computational Creativity (CC) community to make objective, falsifiable claims about progress made from one version of a program to another, or for comparing and contrasting different software systems for the same creative task, we need concrete measures of evaluation. There are two notions of evaluation in CC: (*i*) judgements which determine

F^p:	a method for generating framing information
F^g:	an item of framing information
A^p:	a method for generating aesthetic measures
A^g:	an aesthetic measure
C^p:	a method for generating concepts
C^g:	a concept
E^p:	a method for generating expressions of a concept
E^g:	an expression of a concept

Any particular creative episode can be expressed in terms of at least one of these components (it may well be the case that not all of the components will be present). Note that items of framing information, F^g, may refer to any combination of A^p, A^g, C^p, C^g, E^p, and E^g. In order to cover as many creative acts as possible, we assume only that there must be something new created for the question of creativity to arise. This could be very small, a brush stroke of an artist, an inference step by a mathematician, a single note in a piece of music. Our model, then, covers "merely generative" acts as well as "fundamentally generative" acts. By drawing our baseline at "merely generative", our model can be used to describe the most basic "creative act" possible, we avoid the thorny issue

Aspirations for

Computational Creativity Theory

- Aim for computational learning theory:
  - "To give a rigorous, computationally detailed and plausible account of how learning can be done"

(Dana Angluin)

Aspirations for

Computational Creativity Theory

- Aim for computational creativity theory:
  - "To give a rigorous, computationally detailed and plausible account of how creativity can be done"

Aim is to prove theorems about the nature of software, to enable comparisons

Ground the theory in reality with respect to the amount of resources, user interaction, etc.

Not aiming to capture all senses in which software can create, but be a rallying point

Tiers of words

The theory will be underpinned by the foundational notions of a **creative act** and its **impact**, as described in the FACE and IDEA models above. These will be expanded to allude to the tier 2 aspects, and we will add a model of potential **personality** traits in software, as creative behaviour in software can be influenced by simulation of such traits.

ve will expand the notion of creative act by looking in de ail at **process**, i.e., by formalising aspects of how individ ual generative acts are performed within creative acts. This formalism will also make the impact model more acute, for instance capturing the notion that certain processes ma be perceived as more intelligent (e.g., generalisation, de duction) than others (e.g., random selection, exhaustive earch). In this tier, we will look specifically at **problem solving** processes, as this will enable us to encompass many traditional AI approaches, and appeal to formalisms arising from their study. As most creative programs work in collaboration with people, we need to capture notions of **interaction** in a formal setting, so that attribution of creativity and suggestions for more creative collaborations can be made. In particular, it will be imperative to capture as

ects of the **programming** people undertake to build their reative system, and aspects of how people might form an **interpretation** of a creative act. To address this, we will formalise aspects of how software can have a model of their **audience**, in particular the **knowledge** shared between programmer, software, user and audience members. To further expand all three foundational models, we will explore the notion of **framing** already sketched in the FACE model, by formalising how software, its programmers and its users can add value to the artefacts they generate.

ne baseline models will provide more acute formalisms of pecific aspects of intermediate models. In particular, we will unpack the notions of interpretation and interaction by looking in detail at **affect** and **surprise**, which will further involve addressing formal notions of **context**, **novelty** and **humour** in relation to the audience and knowledge inter mediate models. To add further precision to the interac ion model, we will look at **community** issues that arise, nd derive simplistic but formal notions of **humanity** and its role in Computational Creativity. These notions will in volve aspects of **trust** and **physicality**, both of which can have an impact on the nature and interpretation of creative acts. To add further precision to the interpretation model, we will look at ways in which software can try to control how people assess generated artefacts, for instance via obfuscation. To unpack the problem solving model we

To unpack the problem solving model we ill derive formalisms of **search**, including the notion of **exploration** and **transformation** of search spaces, the use of **analogy** and the role of **metaphor** in creative reasoning, all of which have already been studied extensively in the Computational Creativity literature. We will derive formalism of notions of **idea formation**, including specific aspects of **imagination** and the **blending** of concepts. We will unpack the intermediate framing model by formalising how software can add value to its creations by providing discursive material. This will involve a model of how software can demonstrate an **appreciation** of the creative acts it and others perform, and how it can form and express certain **intentionality**, and demonstrate its **autonomy**. It will also involve models of how software can express **curiosity** and **playfulness**, and use these to increase the cultural value of its creations.

Descriptive Models

Should Provide...

- Some simplifying assumptions related to programming/ running software and the appreciation by an audience of its behaviour and its output
- A set of conceptual definitions which can be used to describe behaviour in software/programmers/audiences associated with acts of creation
- A set of concrete calculations based on the definitions, which can be used to compare and contrast different software systems
- Some suggesions for how the calculations could be applied in different application domains

The FACE model

To describe creative acts performed by software

- Simplifying assumptions:
  - Even the smallest generative act can be described as a creative act (e.g., multiplying two numbers together)
    - Independently of the amount of impact the act might have
  - We can effectively restrict ourselves to discussing how software can produce eight types of output
  - Both the processes performed by software and the results of the processing need to be covered
  - The quality and quantity of creative acts can be used to compare creative software

The FACE model

To describe creative acts performed by software

facts are invented. We define a *creative act* as a non-empty tuple of generative acts. Each tuple contains exactly zero or one instance of eight types of individual generative act. The different types of generative act are denoted by the following letters with g or p superscripts, representing generative acts which produce:

E^g:	an expression of a concept
E^p:	a method for generating expressions of a concept
C^g:	a concept
C^p:	a method for generating concepts
A^g:	an aesthetic measure
A^p:	a method for generating aesthetic measures
F^g:	an item of framing information
F^p:	a method for generating framing information

- We use lower case to denote the output from the individual generative acts in the creative act tuples, and a bar notation to indicate constituent generative acts performed by a third party

The FACE model

To describe creative acts performed by software

- Comparison methods:
  - Volume of creative acts
  - Ordering of creative acts, e.g., <A^g, C^g, E^g> deemed more creative than <C^g, E^g>
  - By the nature of the processes, e.g., random deemed less creative than inductive
- By using the aesthetic function (given or invented) in a domain

$$average(S) = \frac{1}{n}\sum_{i=1}^{n} \overline{a^g}(c_i^g, e_i^g)$$
$$best\_ever(S) = max_{i=1}^{n} (\overline{a^g}(c_i^g, e_i^g))$$
$$worst\_ever(S) = min_{i=1}^{n} (\overline{a^g}(c_i^g, e_i^g))$$
$$precision(S) = \frac{1}{n}|\{(c_i^g, e_i^g) : 1 \leq i \leq n \wedge \overline{a^g}(c_i^g, e_i^g) > t\}|$$
$$reliability(S) = best\_ever(S) - worst\_ever(S)$$

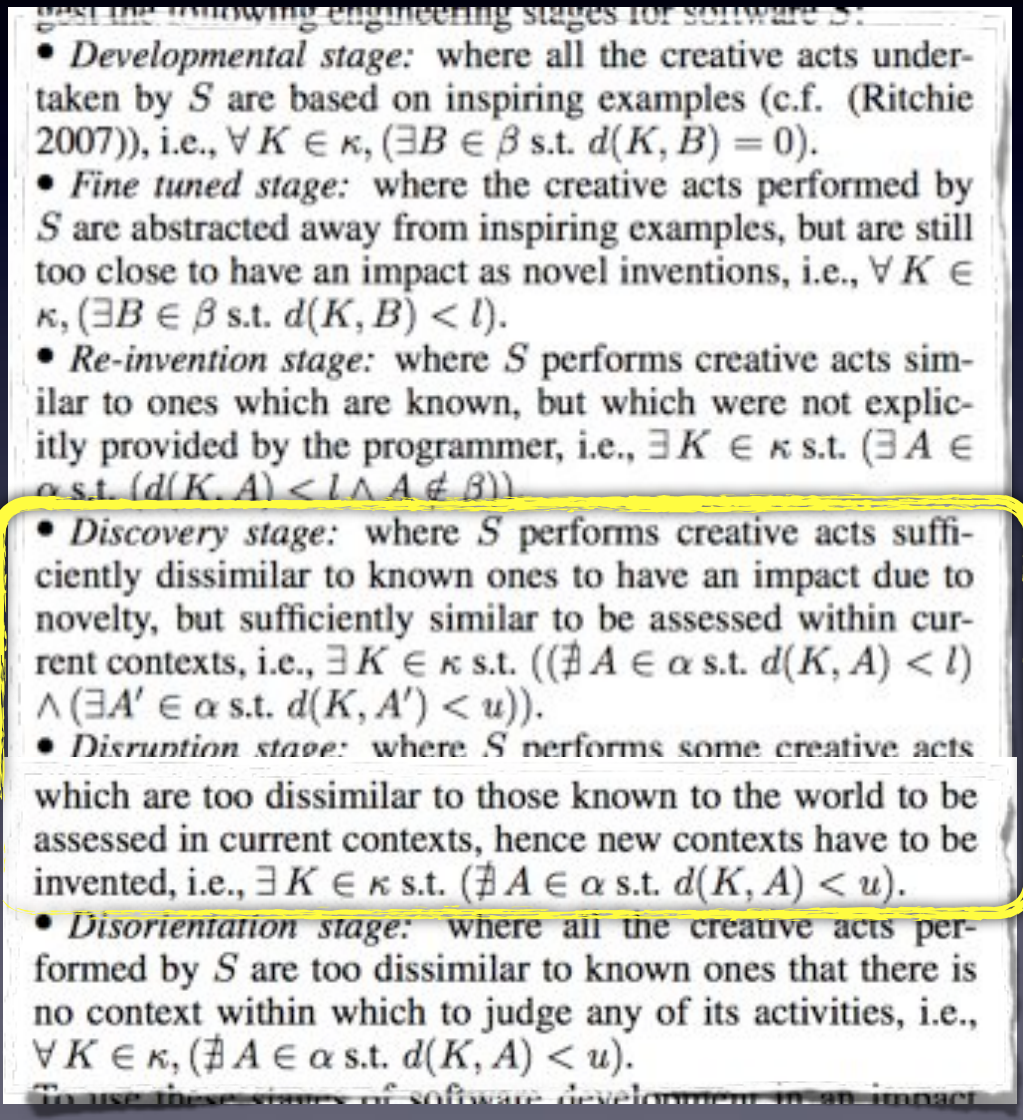
The IDEA model

To describe the impact that creative acts may have

- Motivations
  - Creative software can invent its own aesthetics, so we need to generalise past value judgements
  - The influence of the programmer/user has to be assessed to evaluate the impact caused by the behaviour of the software
- Simplifying assumptions
  - An ideal software development process described by FACE-tuples
  - Full knowledge of the creative acts that went into the production of all the relevant background knowledge
  - An ideal audience of members, m, able to perfectly assess their appreciation of creative acts, A, along two axes:
    - Well being: $wb_m(A)$ and cognitive effort: $ce_m(A)$ [Note not creativity directly]

The IDEA model

To describe the impact that creative acts may have



- *Developmental stage:* where all the creative acts undertaken by S are based on inspiring examples (c.f. (Ritchie 2007)), i.e., $\forall K \in \kappa, (\exists B \in \beta \text{ s.t. } d(K,B) = 0)$.
- *Fine tuned stage:* where the creative acts performed by S are abstracted away from inspiring examples, but are still too close to have an impact as novel inventions, i.e., $\forall K \in \kappa, (\exists B \in \beta \text{ s.t. } d(K,B) < l)$.
- *Re-invention stage:* where S performs creative acts similar to ones which are known, but which were not explicitly provided by the programmer, i.e., $\exists K \in \kappa \text{ s.t. } (\exists A \in \alpha \text{ s.t. } (d(K,A) < l \wedge A \notin \beta))$.
- *Discovery stage:* where S performs creative acts sufficiently dissimilar to known ones to have an impact due to novelty, but sufficiently similar to be assessed within current contexts, i.e., $\exists K \in \kappa \text{ s.t. } ((\nexists A \in \alpha \text{ s.t. } d(K,A) < l) \wedge (\exists A' \in \alpha \text{ s.t. } d(K,A') < u))$.
- *Disruption stage:* where S performs some creative acts which are too dissimilar to those known to the world to be assessed in current contexts, hence new contexts have to be invented, i.e., $\exists K \in \kappa \text{ s.t. } (\nexists A \in \alpha \text{ s.t. } d(K,A) < u)$.
- *Disorientation stage:* where all the creative acts performed by S are too dissimilar to known ones that there is no context within which to judge any of its activities, i.e., $\forall K \in \kappa, (\nexists A \in \alpha \text{ s.t. } d(K,A) < u)$.

- Need a distance function, d, to tell how close two creative acts are
- Formalism for the development of creative software with respect to the programmer/user's influence
- Compare software in terms of its autonomy from the programmer and from the cultural context it was programmed within

The IDEA model

To describe the impact that creative acts may have

$$dis(A) = disgust(A) = \frac{1}{2n}\sum_{i=1}^{n}(1 - wb_i(A))$$
$$div(A) = divisiveness(A) = \frac{1}{n}\sum_{i=1}^{n}|wb_i(A) - m(A)|$$
$$ind(A) = indifference(A) = 1 - \frac{1}{n}\sum_{i=1}^{n}|wb_i(A)|$$
$$pop(A) = popularity(A) = \frac{1}{2n}\sum_{i=1}^{n}(1 + wb_i(A))$$
$$prov(A) = provocation(A) = \frac{1}{n}\sum_{i=1}^{n}(ce_i(A))$$

$$acquired\_taste(A) = (pop(A) + prov(A))/2$$
$$instant\_appeal(A) = (1 + pop(A) - prov(A))/2$$
$$opinion\_splitting(A) = (1 + div(A) - prov(A))/2$$
$$opinion\_forming(A) = (div(A) + prov(A))/2$$
$$shock(A) = (1 + dis(A) - prov(A))/2$$
$$subversion(A) = (dis(A) + prov(A))/2$$
$$triviality(A) = (1 + ind(a) - prov(A))/2$$

- Formalism attempting to capture some common notions of impact, using the well-being and cognitive effort measures of the ideal audience
- m(A) is the mean well being amongst the ideal audience

Comparison Study

Mathematical Discovery Software

- Comparison of types of creative act
  - AM and HR: $\langle \overline{A}^g, C^g, E^g \rangle$
    - But HR has more types of C^g and E^g generative acts
  - Meta-HR: $\langle C^p, C^g, E^g \rangle$ and $\langle A^g, C^g, E^g \rangle$
  - TM took Model Generation from $\langle E^g \rangle$ to $\langle C^g, E^g \rangle$
- In terms of precision, AM outperforms HR, but AM never left the fine-tuned stage of development, whereas we argue that HR is in the discovery stage, hence has had more impact

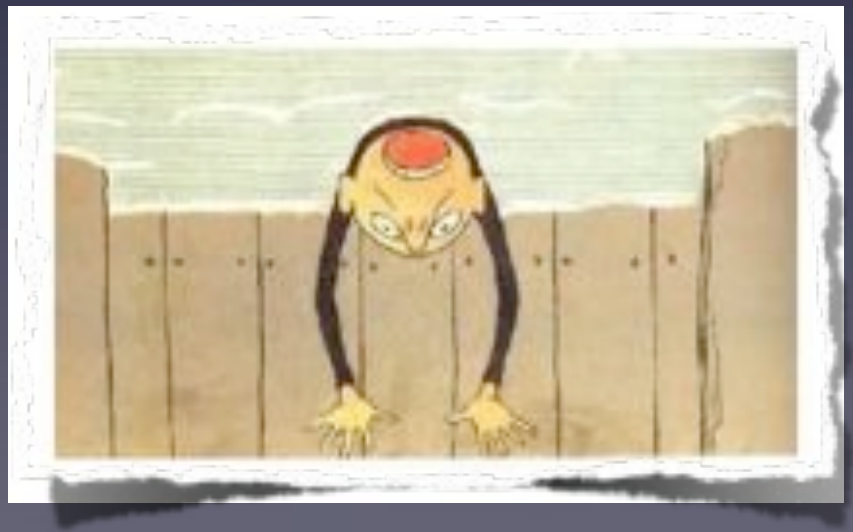
Comparison Study

Art Generation Software

- Comparison of types of creative act
  - AARON and The Painting Fool: $<C^g, E^g>$
    - But The Painting Fool has more types of C^g
  - The Painting Fool collage generation: $<A^g, C^g, E^g>$
  - TPF + HR fitness function invention:
    - $<A^g, C^g, E^g>$ = <fitness function, scene, rendering>
- Most evolutionary art systems: $<\overline{A^g}, C^g, E^g>$, but NEvAr performs creative acts of the form: $<F^g, \overline{A^g}, C^g, E^g>$ because it uses mathematical fitness functions

See Alison's Paper for...

- Motivations for the FACE and IDEA models coming from cognitive science, psychology and philosophy
- Some links to existing Computational Creativity formalisms, such as from Ritchie, Wiggins, etc.
- Case studies from the history of mathematics and the visual arts



The Next Steps

- We're concentrating on formally capturing notions of *progress* in our field
  - And writing up the next version of Computational Creativity Theory
- Diagrammatic model, where each diagram represents aspects of
  - How the software/user work together at runtime in terms of FACE
  - How the software was developed by a programmer
  - The results of evaluating the artefacts produced
- Certain changes in the diagram will map onto our intuitions of progress in Computational Creativity, e.g., the removal of a bar, increase in evaluation
  - This will enable mapping onto categories such as "definite progress", "definite regress", "possible progress", etc.

Final Guidelines

- The idea is to possibly appeal to these guidelines during the engineering, testing and engagement parts of your project

- But also, they're here to get you thinking about some more of the philosophical aspects of Computational Creativity research

1. Ever decreasing circles

- It's important to recognise that we have the potential to contribute as much to the understanding of human creativity as psychological studies do
- We don't necessarily have to wait for discoveries about the nature of human creativity to add creative behaviour to our software
- *AI researchers are the best people to implement creative behaviours in software*
- We can imagine mutual benefits where all fields learn from each other - spiralling down to the truth

2. Paradigms lost

- As AI researchers and practitioners, you don't necessarily have to see every intelligent task as a problem solving exercise
- If you do apply a reductionist approach, remember to put the pieces back together again
- The artefact generation paradigm has been rediscovered: intelligent tasks are framed as opportunities to generate something of cultural value

3. The whole is more than a sum of the parts

- It is often more difficult to get your software to talk to other software than to implement a pale version of the software you want
- However, it's likely that your software will be more powerful if you join forces with others
- And it helps to attract people to Computational Creativity if we use their software

4. Climbing the meta-mountain

- We need to constantly ask ourselves how we can hand over creative responsibilities to the software
- Plan in advance to one day get the software to take over what you are doing in projects
- In particular, think about how the software can take on aethetic responsibilities, and possibly show intentionality in its work
- Try and hand over meta-level control and climb the mountain to the top

5. The creativity ~~tripod~~

- People often take details of a generative process into account when they valuate output artefacts
- The default position in public perception is that software cannot be creative, which can lead to a vicious circle where output is never seen as valuable
    - Hence, we need to manage this public perception
- People will generally not ascribe creativity to software if it is lacking skill, appreciation or imagination. So, we can be proactive and aim to implement behaviours which tick these boxes
- Remember that tripods have three legs, with three sections to each leg: (programmer,/user software, audience)

6. Beauty is in the mind of the beholder

- Value is not just skin-deep
- If you aim for pastiche, you might get useful software, but it's unlikely to ever be taken seriously as creative in its own right
  - And this may well impact how people evaluate the output
- Think about the process/output of/from your software having an impact on people, rather than the imitation game
- Ask yourself: "Is a Turing-style test the right way to assess your software?" - people may to know about the entire creative act if they are to assess the output

7. Good art makes you think

- The output of creative software should really be seen as an invitation to start a dialogue
- Decorative art has value, but it is unlikely to be seen as great art, because it doesn't give people an opportunity to have a dialogue with the artwork
  - Dialogues can be audience-centric, or involve cultural aspects of the day, historical concepts, etc.

- *Our flavour of AI makes people think more rather than less*

The Take Home...

- The discussions you've seen between the lecturers here (in public and in the pub) highlight that this is a great time to get involved in helping us to define aspects of our field through formalisations of notions of creativity in software

- "We can only see a short way ahead, but we can see plenty there that needs to be done"

  Alan Turing

Here's how to get Involved Next

Paper Types

- *Technical papers:* these will be papers posing and addressing hypotheses about aspects of creative behaviour in computational systems
- *System description papers:* these will be papers describing the building and deployment of a creative system value in one or more domains
- *Study papers:* these w uch as psychology, philosophy, cognitive to broader areas of Artificial Intelligence ch appeal to studies of the field of Computa
- *Cultural application p* sage of creative software in a cultura rts/recordings/scores; poetry or story reac sults for scientific journals or scientific practice; released games/game jam entries
- *Position papers:* these will be papers presenting an opinion on some aspect of the culture of Computational Creativity research, including discussions of future directions, past triumphs or mistakes and issues of the day

And think about a late breaking paper!

www.computationalcreativity.net

www.prosecco-network.eu

ccg.doc.ic.ac.uk

Thanks to EPSRC grant EP/J004049, EU projects PROSECCO and WHIM