Computer-Assisted Authoring of Interactive Narratives
Supplementary Document

A Behavior Trees for Narrative Authoring

This section outlines naive approaches to using behavior trees for authoring interactive narratives, highlighting its problems and justifying the need for interactive behavior trees

Monitoring User Interactions. Fig. 1 illustrates a naive approach to monitoring user interactions in a BT. This requires the subtree that monitors user input to be inserted into the narrative tree at all points where user selects how the narrative proceeds, thus producing a tight coupling between the narrative definition and user interaction. Reducing the number of instances in the narrative where user input is monitored severely limits interactivity, while free-form interaction increases the complexity of the tree. Also, user input can be monitored only at the granularity of a single node in the tree, and not at every frame. These complications produce a tradeoff between complexity in the narrative and degree of interactivity.

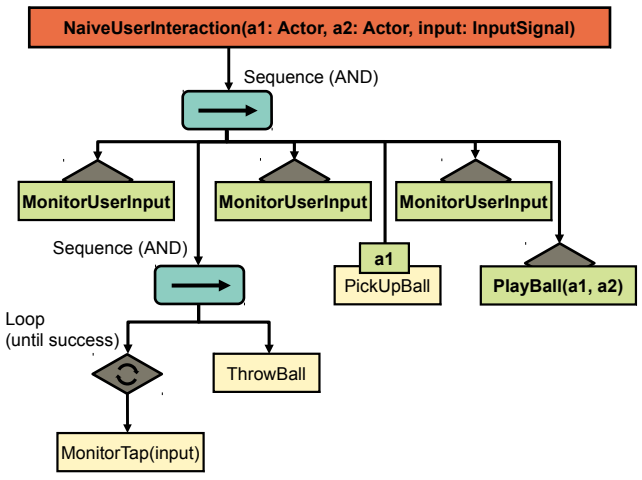


Figure 1: *Handling user interaction in a traditional Behavior Tree. The MonitorUserInput subtree is required at all points in the narrative tree where the user selects how the narrative proceeds, resulting in unnecessary large trees.*

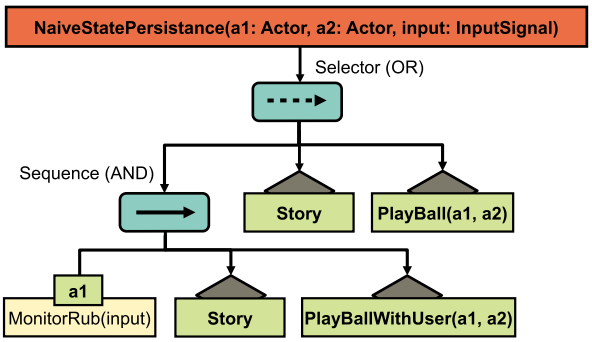


Figure 2: *A naive approach to handle state persistence in Behavior Trees.*

State Persistence. Behavior Trees traditionally don't have any means of explicitly storing the past state of the characters involved in the narrative, where the current state of the story is implicit in the current active node. Fig. 2 illustrates a simple narrative where the bears choose to include the user in the game of catch contingent on whether the user playfully interacted with the bear at the beginning of the story. A traditional BT requires a branch at the very beginning of the narrative where the presence or absence of the playful rub produces two largely redundant tree definitions with minor variations in its ending.

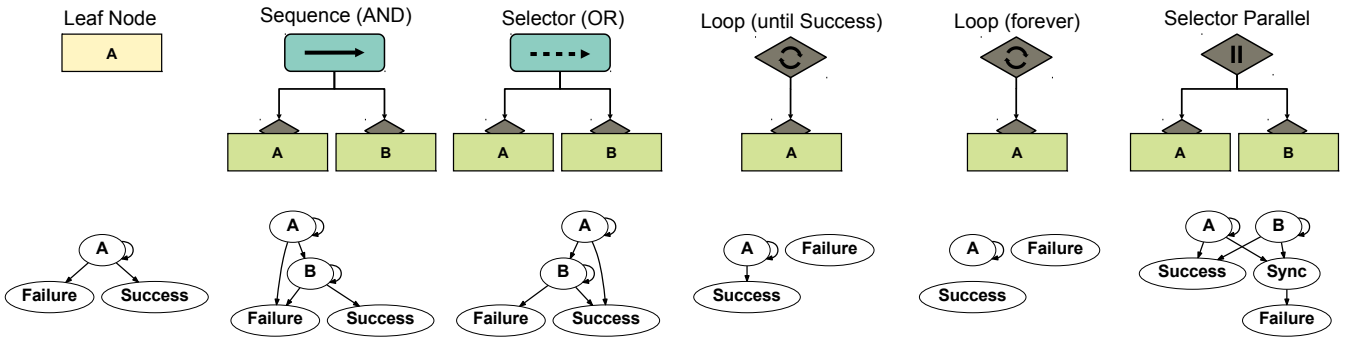
B Belief State Computation

Alg. (1) outlines the algorithm for computing belief states $\mathbf{b}$ for each type of node in the behavior tree using a visitor design pattern. We keep track of a set of failure states $\mathbf{b}^f$ which are potential states that have failed during certain execution paths of the BT leading up to that particular node. For an affordance node, the postconditions of the affordance are simply applied to all states in the belief state. For a condition node, there are three possible cases for each state in the belief state: (1) If the conditions are satisfied, there is no change. (2) If the conditions are not satisfied, the state is moved to $\mathbf{b}^f$. (3) If the condition cannot be evaluated by the partially specified state, we add the needed attribute values to the state and also add the negation of the desired attributes to the set of failed states. The belief states of other control nodes are computed similarly.

C Computing Cyclomatic Complexity for Behavior Trees

Fig. 3 illustrates the equivalent control flow representations for the different control nodes used for defining behavior trees, which are used to compute its cyclomatic complexity. All subsequent calculations of $c(.)$ assuming that the behavior trees are converted to their equivalent control flow graphs.

Leaf Node. A leaf node $\mathbf{t}_{\text{leaf}}$ represents an atomic command in a BT which returns either success or failure. If it is in the "running" state, it continues executing itself until it succeeds or fails. Fig. 3(a) shows the CFG for a leaf node. Depending on the number



(a) Leaf (b) Sequence (c) Selector (d) Loop until success (e) Loop Forever (f) Selector Parallel

Figure 3: *Control flow graphs for the different control nodes used in Behavior Trees.*

of return states in the particular leaf node implementation, we have $p(\mathbf{t}_{\text{leaf}}) = \{0, 1, 2\}$ and $s(\mathbf{t}_{\text{leaf}}) = \{1, 2\}$. If the leaf node immediately returns success or failure without using the running state, we have $c(\mathbf{t}_{\text{leaf}}) = 1$. If the leaf node can enter the running state, the complexity is $c(\mathbf{t}_{\text{leaf}}) = 2$.

Sequence Node. A sequence node $\mathbf{t}_{\text{seq}}$ returns failure if any one of its child nodes fails, else it returns success. If a child returns "running", it simply continues executing this child until it has reached failure or success. Fig. 3(b) illustrates the CFG for $\mathbf{t}_{\text{seq}}$. To calculate $c(\mathbf{t}_{\text{seq}})$ of $\mathbf{t}_{\text{seq}}$ with a set of m child nodes $\{\mathbf{t}_i | \mathbf{t}_1, \mathbf{t}_2 \dots \mathbf{t}_m\}$, we need to consider that each child node may be its own subtree with multiple decision points.

$$p(\mathbf{t}_{\text{seq}}) = \sum_{i=1}^{m} p(\mathbf{t}_i),\ s(\mathbf{t}_{\text{seq}}) = 2,$$
$$\therefore c(\mathbf{t}_{\text{seq}}) = \sum_{i=1}^{m} p(\mathbf{t}_i) \tag{1}$$

Selector Node. The selector node $\mathbf{t}_{\text{sel}}$ returns success as soon as its first child node returns success, and produces a similar control flow graph as compared to $\mathbf{t}_{\text{seq}}$ (Fig. 3(c)). Hence, $c(\mathbf{t}_{\text{sel}}) = c(\mathbf{t}_{\text{seq}})$.

Loop Node. The loop node $\mathbf{t}_{\text{loop}}$ is used to repeatedly execute its child node $\mathbf{t}_{\text{c}}$ until a certain condition is met. A loop node may only have a single decorator or leaf node as its child and does not define how to traverse through multiple children. The following termination conditions may be used: (1) loop until success, (2) loop until failure, (3) loop N times, (4) loop forever. Fig. 3(d) illustrates the loop node which terminates when its child node returns success. It has only one termination node, and an additional decision point is introduced for looping.

$$p(\mathbf{t}_{\text{loop}}) = 1 + p(\mathbf{t}_{\text{c}}),\ s(\mathbf{t}_{\text{loop}}) = 1,$$
$$\therefore c(\mathbf{t}_{\text{loop}}) = p(\mathbf{t}_{\text{c}}) + 2 \tag{2}$$

The same calculations apply for a loop node that repeats until failure. For $\mathbf{t}_{\text{loop}}$ which repeats a fixed number of times.

$$p(\mathbf{t}_{\text{loop}}) = 1 + p(\mathbf{t}),\ s(\mathbf{t}_{\text{loop}}) = 2,$$
$$\therefore c(\mathbf{t}_{\text{loop}}) = p(\mathbf{t}) + 1 \tag{3}$$

Fig. 3(e) illustrates a loop node that never returns. Since the node never terminates and has neither a decision point nor an exit point, we only need to consider the child of the loop node.

$$p(\mathbf{t}_{\text{loop}}) = p(\mathbf{t}),\ s(\mathbf{t}_{\text{loop}}) = 0,$$
$$\therefore c(\mathbf{t}_{\text{loop}}) = c(\mathbf{t}) \tag{4}$$

Parallel Node. The parallel node $\mathbf{t}_{\text{par}}$ executes its child nodes in parallel and has two types depending on the termination condition: (1) Selector Parallel: It executes until any child node returns success or all of them return failure. (2) Sequence Parallel: It executes until any child node returns failure or all of them succeed. Fig. 3(f) illustrates the control flow graph of a selector parallel node. An additional node "Sync" is used to symbolize the synchronization barrier between the child nodes and termination. The sequence parallel node exhibits similar behavior and both their complexity measures can be calculated as shown below.

$$p(\mathbf{t}_{\text{par}}) = 2 \cdot \sum_{i=1}^{m} p(\mathbf{t}_i),\ s(\mathbf{t}_{\text{par}}) = 2,$$
$$\therefore c(\mathbf{t}_{\text{par}}) = 2 \cdot \sum_{i=1}^{m} p(\mathbf{t}_i) \tag{5}$$

C.1 Behavior Tree Naive Approach CC

Behavior Trees (Naive Approach). Fig. 1 illustrates the naive approach to authoring interactive narratives using BT's. This BT $\mathbf{t}_{\text{BT}} = \langle \mathbf{t}_{\text{mu}}, \{a_i | a_1 \dots a_m\} \rangle$ comprises a subtree $\mathbf{t}_{\text{mu}}$ which monitors user input and story branching, and m story arc subtrees, where a_i represents the i^{th} arc with $|a_i|$ nodes. However, these subtrees are tightly coupled together as the user input must be monitored between each node of each story arc to ensure freeform user interaction. The number of decision points for each story arc a_i, $p(a_i) = |a_i| \cdot p(\mathbf{t}_{\text{mu}}) + p(a_i)$, and $\mathbf{t}_{\text{BT}}$ has two exit points. Hence, the cyclomatic complexity $c(\mathbf{t}_{\text{BT}})$ is calculated as follows:

$$c(\mathbf{t}_{\text{BT}}) = \sum_{i=1}^{m} (|a_i| \cdot p(\mathbf{t}_{\text{mu}}) + p(a_i)) \tag{6}$$

Algorithm 1 Belief State Computation

{Affordance Node}
function visit $(\mathbf{t}, \mathbf{b}, \mathbf{b}^f)$
  for all $\mathbf{s} \in \mathbf{b}$ **do**
    $\mathbf{s} = \Omega_{\mathbf{t}}(\mathbf{s})$
  return $\langle \mathbf{b}, \mathbf{b}^f \rangle$

{Condition Node}
function visit $(\mathbf{t}, \mathbf{b}, \mathbf{b}^f)$
  for all $\mathbf{s} \in \mathbf{b}$ **do**
    if $\mathbf{\Phi_t}(\mathbf{s}) ==$ `FALSE` **then**
      $\mathbf{b} = \mathbf{b} - \mathbf{s}$
      $\mathbf{b}^f = \mathbf{b}^f \cup \mathbf{s}$
    else if $\mathbf{\Phi_t}(\mathbf{s}) ==?$ **then**
      {Precondition cannot be evaluated by partial state $\mathbf{s}$}
      $\mathbf{b} = \mathbf{b} - \mathbf{s}$
      $\mathbf{s}_p = \mathbf{s} \wedge \mathbf{\Phi_t}$
      $\mathbf{s}_n = \mathbf{s} \wedge \neg\mathbf{\Phi_t}$
      $\mathbf{b} = \mathbf{b} \cup \mathbf{s}_p$
      $\mathbf{b}^f = \mathbf{b}^f \cup \mathbf{s}_n$
  return $\langle \mathbf{b}, \mathbf{b}^f \rangle$

{Sequence Node}
function visit $(\mathbf{t}, \mathbf{b}, \mathbf{b}^f)$
  for all $\mathbf{t_c} \in \mathbf{t}$ **do**
    visit$(\mathbf{t_c}, \mathbf{b}, \mathbf{b}^f)$
  return $\langle \mathbf{b}, \mathbf{b}^f \rangle$

{Selector Node}
function visit $(\mathbf{t}, \mathbf{b}, \mathbf{b}^f)$
  $\mathbf{b}_t = \mathbf{b}$
  $\mathbf{b} = \emptyset$
  for all $\mathbf{t_c} \in \mathbf{t}$ **do**
    $\langle \mathbf{b}_c, \mathbf{b}_c^f \rangle =$ **visit**$(\mathbf{t_c}, \mathbf{b}_t, \mathbf{b}^f)$
    $\mathbf{b} = \mathbf{b} \cup \mathbf{b}_c$
    $\mathbf{b}_t = \mathbf{b}_c^f$
  return $\langle \mathbf{b}, \mathbf{b}^f \rangle$

{Loop Node}
function visit $(\mathbf{t} = \langle r, \mathbf{t_c} \rangle, \mathbf{b}, \mathbf{b}^f)$
  if $\mathbf{t_c} \neq$ `null` **then**
    $\mathbf{b}_c =$ **visit**$(\mathbf{t_c}, \mathbf{b}, \mathbf{b}^f)$
    if $r \neq$ `RepeatUntilSuccess` **then**
      $\mathbf{b} = \mathbf{b} \cup \mathbf{b}_c$
  return $\langle \mathbf{b}, \mathbf{b}^f \rangle$

{Parallel Node}
function visit $(\mathbf{t}, \mathbf{b}, \mathbf{b}^f)$
  if `SelectorParallel` **then**
    same as for selector node
  else
    same as for sequence node
  return $\langle \mathbf{b}, \mathbf{b}^f \rangle$

D Augmented Reality Framework

Augmented Reality (AR) applications benefit from intuitive and versatile input mechanisms where the user can seemingly physically interact with the virtual world. For see-through AR applications, the physical movement of the mobile device serves as a direct means of exploring the digital content superimposed in a real environment, where, not only can the user interact with the virtual characters using the host of sensors available on the device, but the virtual characters can interact with the physical world as well.

Implementation. We used a natural image-based tracking approach [Ababsa and Mallem 2004] which registers the 2D marker image and estimates the camera location and pose in real-time for stable tracking. We used the implementation provided by Vuforia [Qualcomm 2010]. Additional image markers can also be tracked and used as triggers in the AR application, for example, to instantiate new objects in the world, which may branch the narrative in a different direction. The game application was implemented using the Unity3D game engine using a data-driven character animation system. The animation functionality is exposed to the author using a set of routines including **LookAt**(obj), **Reach**(target), **Reach**(target) etc. which can be invoked from BT's. For more details of the animation system, please refer to [Shoulson et al. 2013]. The narrative was authored using an extended version of the BT library described in [Shoulson et al. 2011]. The current version of the game was deployed and tested on multiple portable devices including Sony Xperia Tablet Z, Apple iPad 3rd generation, Apple iPad Mini Retina, and Apple iPad Air.

Interaction Vocabulary. Using the sensors available on the mobile device, the user can interact with the virtual characters in the following ways: (1) Moving the device to focus on different objects and characters. (2) Tapping on objects and characters to pick them up or interact with them. (3) Shaking the device. (4) Gestures to communicate user intent. (5) Using image markers to trigger objects (e.g., a honey pot sticker can be used to create a honey pot for the bears) in the world. The mode and effect of interactions is completely decoupled from the narrative and can be easily changed depending on the platform used without impacting the narrative definition.

E Scenario Definition

Table 1 outlines representative affordances that were used to author the interactive narratives described in the main document.

F User Study Data

Table 2 provides the raw data that was collected during the user study for reference.

References

ABABSA, F.-E., AND MALLEM, M. 2004. Robust camera pose estimation using 2d fiducials tracking for real-time augmented reality systems. In *Proceedings of the 2004 ACM SIGGRAPH International Conference on Virtual Reality Continuum and Its Applications in Industry*, ACM, New York, NY, USA, VRCAI '04, 431–435.

QUALCOMM, 2010. Vuforia Developer SDK.

SHOULSON, A., GARCIA, F. M., JONES, M., MEAD, R., AND BADLER, N. I. 2011. Parameterizing behavior trees. In *Proceedings of the 4th International Conference on Motion in Games*, Springer-Verlag, Berlin, Heidelberg, MIG'11, 144–155.

SHOULSON, A., MARSHAK, N., KAPADIA, M., AND BADLER, N. I. 2013. Adapt: the agent development and prototyping testbed. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, I3D '13, 9–18.

Affordance *IsInfatuated*(Character c):
Precondition Φ:
InScene(self) ∧ InScene(c);
Postcondition Ω:
InfatuatedWith(self, c);

(a)

Affordance *WinLove*(Character c, Ball b):
Precondition Φ:
InScene(self) ∧ InScene(c)
∧ HoldsBall (self, b)
∧ InLove(self, c)
∧ EnchantedBy(c, b);
Postcondition Ω:
¬Holds (self, b)
∧ Holds (c, b)
∧ InLove (c, self);

(b)

Affordance *Flirt*(Character c):
Precondition Φ:
InScene(self) ∧ InScene(c)
∧ InfatuatedWith(self, c);
Postcondition Ω:
EnchantedBy(c, self);

(c)

Affordance *TryToBuyBall*(Vendor v, Ball b):
Precondition Φ:
InScene(self) ∧ InScene(v)
∧ ¬HasMoney(self) ∧ ¬InScene(b);
Postcondition Ω:
NeedsMoney(self);

(d)

Affordance *GetMoney*(Treasure t):
Precondition Φ:
¬HasMoney(self) ∧
¬NeedsMoney(self);
Postcondition Ω:
HasMoney(self);

(e)

Affordance *BuyBall*(Vendor b, Ball b):
Precondition Φ:
InScene(self) ∧ InScene(vendor)
∧ Holds(v,b) ∧ HasMoney(self);
Postcondition Ω:
Holds(self,b) ∧ ¬ Holds(v,b);

(f)

Affordance *Greet*(Character c):
Precondition Φ:
InScene(self) ∧ InScene(c)
∧ ¬Panic(self) ∧ ¬Panic(c);
Postcondition Ω:
Knows(self, c) ∧ Knows(c, self);

(g)

Affordance *GiveBall*
(Character c, Ball b):
Precondition Φ:
InScene(self) ∧ InScene(c)
∧ ¬Panic(self) ∧ ¬Panic(c)
∧ Holds(self, b) ∧ ¬Holds(c, b);
Postcondition Ω:
¬Holds(self, b) ∧ Holds(c, b);

(h)

Affordance *ThrowBall*
(Character c, Ball b):
Precondition Φ:
InScene(self) ∧ InScene(c)
∧ InScene(b) ∧ ¬Panic(self)
∧ ¬Panic(c) ∧ Holds(self, b)
∧ ¬Holds(c, b);
Postcondition Ω:
¬Holds(self, b) ∧ Holds(c, b);

(i)

Affordance *ConverseHappily*
(Character c):
Precondition Φ:
InScene(self) ∧ InScene(c)
∧ ¬Panic(self) ∧ ¬Panic(c)
∧ Knows(self, c)∧ Knows(c, self);
Postcondition Ω:
IsHappy(self) ∧ IsHappy(c);

(j)

Affordance *Argue*(Character c):
Precondition Φ:
InScene(self) ∧ InScene(c)
∧ ¬Panic(self) ∧ ¬Panic(c)
∧ Knows(self, c)∧ Knows(c, self);
Postcondition Ω:
¬IsHappy(self) ∧ ¬IsHappy(c);

(k)

Affordance *AskForBall*
(Character c, Ball b):
Precondition Φ:
InScene(self) ∧ InScene(c)
∧ ¬Panic(self) ∧ ¬Panic(c)
∧ Knows(self, c)∧ Knows(c, self)
∧ ¬Holds(c,b);
Postcondition Ω:
AskedBall(self, c) ∧ Wants(self, b);

(l)

Affordance *PickUp*(User u):
Precondition Φ:
InScene(self) ∧ ¬Holds(u, self);
Postcondition Ω:
IsAttached(self) ∧ Holds(u, self);

(m)

Affordance *ThrowBallToUser*
(User u, Ball b):
Precondition Φ:
InScene(self) ∧ ¬ Panic(self)
∧ InScene(b) ∧ Holds(self, b);
Postcondition Ω:
¬InScene(b) ∧ ¬ Holds(self, b);

(n)

Affordance *PickUpBall*(Character c):
Precondition Φ:
InScene(self) ∧ InScene(c)
∧ ¬IsAttached(self)
∧ ¬Holds(c, self);
Postcondition Ω:
IsAttached(self) ∧ Holds(c, self);

(o)

Affordance *Drop*(Character c):
Precondition Φ:
InScene(self) ∧ InScene(c)
∧ Holds(c, self);
Postcondition Ω:
¬IsAttached(self)
∧ ¬Holds(c, self);

(p)

Affordance *EatHoney*(Character c):
Precondition Φ:
InScene(self) ∧ InScene(c);
Postcondition Ω:
IsHappy(c);

(q)

Affordance *Drop*(User u):
Precondition Φ:
InScene(self) ∧ Holds(u, self);
Postcondition Ω:
¬IsAttached(self) ∧ ¬Holds(u,
self);

(r)

Affordance *FlyToFlower*(Flower f):
Precondition Φ:
InScene(self) ∧ InScene(f);
Postcondition Ω:
AtFlower(self);

(s)

Affordance *AskUserForBall*
(Character c, Ball b):
Precondition Φ:
InScene(c) ∧ ¬InScene(b)
∧ ¬Panic(c) ∧ WantsBall(c, b);
Postcondition Ω:
InScene(b) ∧ ¬WantsBall(c, b);

(t)

Affordance *TakeFromCharacter*(Character
c, Object b):
Precondition Φ:
InScene(c) ∧ InScene(b)
∧ Holds(c, b);
Postcondition Ω:
¬ InScene(b) ∧;

(u)

Affordance *WaveAtUser*(User u):
Precondition Φ:
InScene(self) ∧ Panic(self);
Postcondition Ω: none

(v)

Affordance *DismissUser*(User u):
Precondition ϕ:
InScene(self) ∧ Panic(self);
Postcondition δ: none

(w)

Affordance *ShakeHead*(User u):
Precondition Φ:
InScene(self) ∧ Panic(self);
Postcondition Ω: none

(x)

Affordance *StopBeePanic*(Bees b):
Precondition Φ:
InScene(self) ∧ InScene(b)
∧ AtFlower(b);
Postcondition Ω:
¬Panic(self);

(y)

Affordance *StartBeePanic*(Bees b):
Precondition Φ:
InScene(self) ∧ InScene(b);
Postcondition Ω:
Panic(self);

(z)

Affordance *Pet*(User u):
Precondition Φ:
InScene(self);
Postcondition Ω:
IsHappy(self);

(α)

Table 1: *Overview of representative affordances used to author the interactive narrative. The affordance owners (self) are not shown as parameters in the affordance definition for simplicity.*

Table 2: *Measured raw data from user study*

Proficiency	Method	Time to Author t_a	Number of Bugs n_b	Subjective Difficulty d_s	Number of Clicks n_c	User ID
expert	SG	25	526	5	964	12
expert	IBT	6	11	3	239	12
expert	IBTA	5	0	2	263	12
expert	SG	53	234	5	1412	11
expert	IBT	10	72	3	392	11
expert	IBTA	6	0	1	304	11
expert	SG	49	145	5	584	10
expert	IBT	19	45	3	333	10
expert	IBTA	15	0	1	263	10
novice	SG	145	456	4	1432	9
novice	IBT	101	230	3	981	9
novice	IBTA	63	0	1	441	9
novice	SG	123	443	5	1543	8
novice	IBT	75	124	3	894	8
novice	IBTA	45	2	2	680	8
novice	SG	100	230	5	987	7
novice	IBT	78	187	3	785	7
novice	IBTA	54	0	2	568	7
novice	SG	110	560	5	1430	6
novice	IBT	89	334	3	1023	6
novice	IBTA	56	0	1	680	6
novice	SG	98	552	5	889	5
novice	IBT	89	243	3	803	5
novice	IBTA	47	0	2	432	5
expert	SG	53	187	5	584	4
expert	IBT	37	23	3	125	4
expert	IBTA	19	0	1	108	4
novice	SG	149	38	5	513	3
novice	IBT	93	2	0	812	3
novice	IBTA	61	0	0	1368	3
expert	SG	77	345	5	789	2
expert	IBT	33	24	3	565	2
expert	IBTA	17	12	2	230	2
expert	SG	92	234	5	1021	1
expert	IBT	45	43	2	564	1
expert	IBTA	34	0	1	343	1